UN BUSCADOR INTEL·LIGENT DE VIATGES PER A L'ARGENTINA

AUTOR

JOSÉ ÁNGEL CASADO PÉREZ

Grau en Enginyeria Informàtica - Computació

DIRECTOR

MIQUEL SÀNCHEZ-MARRÈ

Departament de Ciències de la Computació

29 DE JUNY DE 2016

RESUM

Avui dia gairebé la totalitat de les persones que decideixen sortir de viatge busquen i compren els seus bitllets per internet. Un percentatge molt alt d'aquesta gent, que augmenta any rere any, es decanta per la utilització dels serveis que ofereixen les *Travel Search Engines* com *Skyscanner* o *Kayak*, buscadors web que permeten als usuaris trobar les combinacions possibles entre la seva ciutat d'origen i de destí, podent comprovar així quina és la combinació de viatges més econòmica o que més s'adapta a les seves necessitats.

La gran majoria d'aquests buscadors es centren en la cerca de combinacions només tenint en compte el transport amb avió, tot i que comencen a aparèixer al mercat, i amb molta força, altres plataformes, com *Rome2Rio* o *GoEuro*, que permeten trobar combinacions mitjançant més tipus de transports, com tren, autobús o ferri.

En aquest projecte es presenta una solució innovadora per a la construcció d'una *Travel Search Engine* que combini el transport tant d'autobús com d'avió, amb l'objectiu de millorar el transport intern a l'Argentina, un dels països més visitats d'Amèrica del Sud. La solució que es proposa fa una adaptació del paradigma d'Intel·ligència Artificial CBR (*Case-Based Reasoning*, o Raonament Basat en Casos) i el combina amb algorismes de cerca de camins dins d'un graf dirigit amb pes a les arestes, amb l'objectiu de retenir coneixement a partir de les peticions que realitzin els usuaris i millorar així la qualitat de les solucions retornades a aquests.

Els resultats de les proves aplicades al sistema dissenyat, no només mostren com aquest troba combinacions factibles per a les cerques dels usuaris, sinó que a més millora el seu preu i la quantitat, a mesura que s'emmagatzema més informació a la base de dades a partir de l'experiència.

ABSTRACT

Nowadays almost all the people that decide to travel search and buy their tickets online. A large percentage of these people, which increases year after year, choose to use the services provided by *Travel Search Engines* such as *Skyscanner* or *Kayak*. These are web search engines that allow users to find the possible combinations between their origin and destination cities, therefore being able to choose the cheapest of the combinations or the one that best fits their needs.

Most of these search engines focus on searching combinations only for air travel, though there are some new ones, getting strength in the market, such as *Rome2Rio* or *GoEuro*, which allow users to find combinations using more types of transportation, such as train, bus or ferry.

In this project we present a novel approach for the construction of a *Travel Search Engine* that combines both bus and plane transportation, aiming to improve the transportation within Argentina, one of the most visited countries in South America. The solution that is proposed makes an adaptation of the AI paradigm CBR (*Case-Based Reasoning*) and combines it with weighted DAG path searching algorithms, in order to retain knowledge from user queries and therefore improve the quality of the returned solutions.

The results of the applied tests to the designed system don't just show how it finds feasible combinations for the user queries, but also how it improves their price and quantity as it stores more information in the database from experience.

RESUMEN

Hoy en día casi todas las personas que deciden salir de viaje buscan i compran sus billetes por internet. Un porcentaje muy alto de esta gente, que aumenta año tras año, se decanta por la utilización de los servicios que ofrecen las *Travel Search Engines* como *Skyscanner* o *Kayak*, buscadores web que permiten a los usuarios encontrar las combinaciones posibles entre la ciudad de origen y la de destino, pudiendo comprobar así cuál es la combinación de viajes más económica o que más se adapta a sus necesidades.

La gran mayoría de estos buscadores se centran en la búsqueda de combinaciones sólo teniendo en cuenta el transporte en avión, aunque comienzan a aparecer en el mercado, y con mucha fuerza, otras plataformas, como *Rome2Rio* o *GoEuro*, que permiten encontrar combinaciones utilizando otros tipos de transporte, como tren, autobús o ferri.

En este proyecto se presenta una solución innovadora para la construcción de una *Travel Search Engine* que combine tanto el transporte en autobús como en avión, con el objetivo de mejorar el transporte interno en Argentina, uno de los países más visitados de América del Sur. La solución que se propone hace una adaptación del paradigma de Inteligencia Artificial CBR (*Case-Based Reasoning*, o Razonamiento Basado en Casos) y lo combina con algoritmos de búsqueda de caminos dentro de un grafo dirigido con peso en las aristas, con el objetivo de retener conocimiento a partir de las peticiones que realicen los usuarios y mejorar así la calidad de las soluciones devueltas a estos.

Los resultados de las pruebas aplicadas al sistema diseñado, no sólo muestran como este encuentra combinaciones factibles para las búsquedas de los usuarios, sino que además mejora su precio y la cantidad a medida que se almacena más información en la base de datos a partir de la experiencia.

ÍNDEX DE CONTINGUTS

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

ÍNDEX DE DIAGRAMES

1. INTRODUCCIÓ

1.1 CONTEXTUALITZACIÓ

El món de l'*e-torusim*[1] es va començar a concebre en el transcurs dels anys 80 i va esdevenir una realitat tangible a l'última dècada del segle XX. En aquell moment, empreses del sector del turisme (aerolínies, agències de viatges, etc.) van començar a oferir pàgines web pròpies des d'on els clients poguessin trobar informació i reservar els serveis oferts per les mateixes.

Paral·lelament, i curiosament amb una corba de creixement bastant similar, al món de la Intel·ligència Artificial va començar a sorgir i desenvolupar-se un nou paradigma per resoldre problemes (com tants d'altres en aquell moment) basat en la reutilització de coneixement: CBR (*Case-Based Reasoning* o Raonament Basat en Casos). La idea principal d'aquest paradigma és emmagatzemar solucions concretes a problemes ja resolts, que en un futur puguin servir per resoldre problemes de característiques similars.

A començament de segle, van començar a sorgir noves maneres d'*e-tourism* que pretenien donar un ventall de possibilitats encara més ampli als usuaris: les *OTAs*[2] i les *Metasearch Engines*. Les primeres, com *Expedia* o *Orbitz*, no van ser molt més que agències de viatges, però online. Els segons, són webs basats en la cerca exhaustiva a pàgines de tercers, amb la finalitat de sintetitzar i filtrar la informació per a que el client pugui escollir la més adient. Va ser l'aparició d'aquestes noves plataformes el que va propiciar la unió entre l'*e-tourism* i CBR: davant d'una quantitat d'informació tan gran, és inviable buscar per cada petició de l'usuari la solució òptima, sinó que cal basar-se en solucions passades i adaptar-les com convingui. Això sembla que fan webs com *Kayak, Skyscanner*, o *Rome2Rio*, buscadors d'itineraris de viatges (*Travel Search Engines*) que permeten a l'usuari buscar la manera més econòmica o ràpida d'arribar al destí desitjat, aprofitant cerques de clients anteriors.

[1] Turisme electrònic
[2] *Online Travel Agencies*

Aquest projecte pretén crear una *Travel Search Engine* específica per a viatjar dins l'Argentina, que combini tant avió com bus, utilitzant tècniques basades en CBR. Per què l'Argentina? Perquè és un país on el turisme (tant local com estranger) està a l'alça, i no hi ha cap plataforma que doni un suport integral com el que es pretén. A més, l'estat semi-autàrquic dels seus serveis de transports, fa que no s'hagin de consultar desenes d'aerolínies o pàgines amb informació de busos, minimitzant el principal *overhead* d'aquests buscadors: l'adquisició d'informació. I per què tant bus com avió? Si bé és cert que no és un concepte molt explotat (a Europa, només existeixen dues plataformes que ofereixen la cerca amb múltiples mitjans de transport, *GoEuro*[3] y *Rome2Rio*), a països on les vies ferroviàries no han tingut mai suficient inversió (com Argentina, i Sud-Amèrica en general), el transport en bus és el més utilitzat per a viatges de mitja o fins i tot llarga distància.

Com a nota final, cal dir que als últims 10 anys han aparegut altres *Travel Search Engines* no basades en tècniques d'aprenentatge, com *ItaSoftware* o *Farecast*, que basen els seus algorismes en el complex càlcul darrere de les tarifes dels bitllets d'avió. Tot i això, les *Metasearch* com *Kayak* o *Skyscanner* son el tipus de plataformes que s'estan imposant al mercat des de fa uns quants anys enrere. [4]

[3] De fet, a Europa comença a ser un tema interessant, degut a l'aparició de noves companyies *low-cost* d'autobús.

1.2 ELS OBJECTIUS

Vistes les característiques del projecte i quina és la situació general dins la disciplina que es vol tractar, hem d'establir quins són els objectius concrets del nostre projecte.

En general, el projecte tindrà els següents objectius a assolir:

1. Estudiar i implementar el millor sistema informàtic per a un *Travel Search Engine* (*Metasearch*) mitjançant CBR (*back-end*). Això, inclouria:
    a. Estudiar la millor manera d'enregistrar la informació sobre solucions obtingudes dins la base de dades.
    b. Trobar la manera més adequada d'utilitzar les solucions passades per a generar noves solucions.
    c. Trobar el sistema més eficient per a accedir a la informació de terceres webs per tenir informació actualitzada.
2. Dissenyar i implementar una interfície gràfica web el més intuïtiva possible per a un usuari qualsevol (*front-end*).

3. Realitzar una campanya de màrqueting a Internet, i principalment a les xarxes socials, per donar a conèixer la plataforma.

Aquests objectius són part d'un objectiu global, que és facilitar als viatgers de l'Argentina una manera simple de trobar el millor sistema d'anar d'una ciutat a una altra, depenent de les seves prioritats (temps, estalvi...), a partir de tot el ventall de possibilitats que se'ls ofereixi.

A més, la plataforma haurà de ser capaç de mostrar els resultats d'una manera ràpida (com a molt uns 10 segons per a mostrar els primers resultats, i uns 30 per a mostrar-los tots), per a que l'usuari no s'impacienti.

1.3 ACTORS IMPLICATS

Els següents actors són els implicats projecte:

- El desenvolupador, dissenyador i beta tester, que seré jo mateix.
- El director del projecte, en Miquel Sànchez-Marrè, que serà l'encarregat de supervisar-lo i si convé, d'ajudar al desenvolupador en les diferents etapes.

Com a usuaris finals del projecte, distingim els següents tipus:

- Viatgers a l'Argentina que vulguin viatjar amb avió o bus d'una manera econòmica.
- Tercers desenvolupadors que es vulguin aprofitar de la síntesi d'informació que proporcionarà el projecte per a la realització d'altres.

2. ESTAT DE L'ART

2.1 LA SOLUCIÓ INTUÏTIVA: ALGORISMES DE CERCA

Els algorismes de cerca tenen per objectiu trobar, dins d'un graf dirigit amb pes a les arestes, el camí més curt entre un node i un altre, és a dir, permeten trobar el camí la suma dels pesos del qual sigui la mínima. Dins d'aquests algorismes es distingeixen dos tipus: els que realitzen una cerca exhaustiva sobre el graf, on s'exploren els nodes fins a trobar el node de destí només tenint en compte el pes de les arestes, com l'algorisme de *Dijkstra*; i els que realitzen una cerca dirigida, basant-se, a més de en el pes de les arestes, en informació externa – basada, en general, en coneixement expert sobre el domini del problema – que estimi el cost per arribar d'un node al destí, com l'algorisme d'*A**.

Davant un problema com el nostre, en el que hem de trobar la manera menys costosa econòmicament d'anar d'un punt A a un punt B, la solució més intuïtiva, ja que és la que ens donaria el camí òptim entre els dos punts (o els *k* camins òptims, que és el que ens interessaria, per donar totes les possibilitats al usuari), és utilitzar aquests algorismes de cerca, com *Dijkstra* o *A**, on cada ciutat suposaria un vèrtex al graf, i cada trajecte entre una ciutat i una altra, una aresta amb el preu com a pes.

Tot i així, en l'àmbit de la cerca de viatges, aquests algorismes no són massa utilitzats. El principal motiu és l'alta connectivitat dels grafs: com més connexions tingui un graf (més viatges entre ciutats), més ventall de possibilitats hi haurà a l'hora de buscar camins. A més, com més ampla fem la finestra de temps (si no limitem el temps que el viatger vol estar viatjant), aquest ventall de possibilitats es quadruplica [1].

A més del problema de la complexitat temporal de l'algorisme, s'ha d'afegir el problema de mantenir el graf actualitzat. Per a que aquest fos completament consistent (tingués la disponibilitat o els preus actualitzats), el graf s'hauria de generar cada cop que un usuari fes una

consulta; això, tenint en compte que la informació s'ha de extreure d'altres webs, seria pràcticament impossible.

2.1.1 ALGORISME DE DIJKSTRA

L'algorisme de *Dijkstra*, com s'ha comentat anteriorment, executa una cerca exhaustiva o no informada sobre el graf amb l'objectiu de trobar els camins mínims entre un node d'origen i qualsevol altre dins un graf dirigit amb pes a les arestes, i en concret entre un node d'origen i un de destí. El funcionament general de l'algorisme és el següent: aquest manté una cua de prioritat, on va emmagatzemant els nodes trobats fins al moment ordenats per la seva distància al node d'origen (nodes *oberts* per l'algorisme); a cada iteració del bucle principal, s'extreu el mínim d'aquesta cua de prioritat i es *tanca*, és a dir, s'obren els seus fills i s'afegeixen a la cua de prioritat amb la distància de l'origen fins al pare més el cost de l'aresta del pare fins al fill en qüestió. El procés finalitza quan s'han tancat tots els nodes (en cas que es busqui la distància a tots els nodes) o quan es tanca el node de destí (en cas que es busqui la distància a un node en concret). El cost temporal de la implementació més eficient coneguda per l'algorisme [11] és de $O(m + n\log n)$, on m i n són les arestes i els nodes del graf, respectivament.

2.1.2 ALGORISME D'A*

Un altre dels algorismes més populars per cercar el camí òptim entre dos nodes a un graf és *A**. Aquest algorisme, com s'ha dit, realitza una cerca dirigida tenint en compte, a més de la distància al node d'origen que ja utilitzava Dijkstra, una estimació del cost mínim per arribar des d'un node qualsevol fins al node de destí (llavors, evidentment, no és tant comú utilitzar-lo per buscar la distància des d'un origen fins a tots els nodes, sinó més bé fins a un node destí concret). L'estructura algorísmica fonamental és semblant a la de Dijkstra, amb la diferència de que, a l'hora d'ordenar els nodes a la cua de prioritat es té en compte també aquesta estimació donada per una funció heurística, i s'ordenen segons la suma de la distància de l'origen al node i el valor retornat per l'esmentada heurística del node fins al destí. Si l'heurística és coherent amb la

realitat del graf (és a dir, si l'estimació heurística és igual o propera a la distància mínima entre un node donat i el de destí) la complexitat temporal de l'algorisme serà polinòmica i molt millor que la de Dijkstra; en canvi, si l'heurística no es correspon a aquesta distància mínima, es poden *reobrir* nodes tancats anteriorment, fent que el cost de l'algorisme pugui arribar a ser exponencial en el nombre de nodes.

2.1.3 ALGORISMES PER CERCAR ELS *K* CAMINS ÒPTIMS

Tot i que semblen problemes amb mecàniques similars, la cerca dels *k* camins òptims dins un graf dirigit no és tan simple com la cerca d'un únic camí comentada als dos algorismes anteriors. Varis algorismes s'han proposat des dels anys 70 per resoldre el problema en qüestió. El primer d'ells va ser l'*algorisme de Yen* (Jin Y. Yen, 1971), que afrontava el problema adaptant l'algorisme de Dijkstra, assolint una complexitat temporal de $O(kn(m + n\log n))$, on *n* i *m* són, respectivament, els nodes i les arestes del graf, i *k* el nombre de camins a buscar; tot i això, aquest algorisme no permetia l'aparició de cicles al graf. Dècades més tard, David Eppstein va proposar un nou algorisme que sí permetia l'aparició de cicles al graf [10], i a més reduïa la complexitat temporal fins a $O(m + n\log n + k)$; el seu algorisme és, encara avui dia, considerat el referent per resoldre el problema dels *k* camins òptims. Tot i això, el 2011 es va publicar l'algorisme *K** [9], que es basava en la mecànica emprada per l'anterior, però aplicant tècniques de cerca dirigida mitjançant l'algorisme d'A*; aquest nou algorisme, tot i que mantenia una complexitat temporal igual a la de l'algorisme d'Eppstein, va demostrar que aplicant una estimació heurística correcta arribava a superar els temps d'aquest, i a més a més, proporcionava la important qualitat de no requerir mantenir el graf complet a memòria, cosa que sí necessitava l'algorisme anterior. Aquest últim algorisme, que és el que s'utilitzarà al projecte i la funcionalitat concreta del qual es comentarà a l'apartat 6, combina la cerca d'A*, per anar descobrint noves zones al graf, i l'algorisme de Dijkstra, per anar explorant un segon graf creat a partir de la part explorada del graf original, on cada nou node tancat suposarà el següent camí òptim entre l'origen i el destí.

2.2 UNA POSSIBLE SOLUCIÓ: RAONAMENT BASAT EN CASOS

Com s'ha dit abans, la majoria de buscadors de viatges més populars a Internet [4], com *Skyscanner* o *Rome2Rio*, sembla que utilitzen mètodes d'aprenentatge basant-se en les cerques d'usuaris anteriors. Analitzant el funcionament d'aquestes pàgines web, sembla ser que emmagatzemen respostes a cerques anteriors d'usuaris per a que, a cerques similars posteriors, puguin tenir idees generals respecte a quina aerolínia consultar, quines ciutats solen ser ideals per a fer escales, etc. [2] [3] En general, informació que els permeti realitzar una cerca més eficient a les webs dels proveïdors. Aquests mètodes són similars als del paradigma de CBR (Raonament Basat en Casos), que a continuació explicarem.

2.2.1 RAONAMENT BASAT EN CASOS

Com ja s'ha esmentat, CBR és un paradigma de resolució de problemes pertanyent a la Intel·ligència Artificial, fonamentat en la concepció del coneixement humà proposada per Schank i Abelson [7], on la informació relativa a situacions viscudes és emmagatzemada en *scripts*. La premissa principal darrere de CBR és que recordar, entendre, experimentar i aprendre no poden separar-se l'un de l'altre, i que una memòria dinàmica (com la humana) canvia a través de noves experiències, adaptant el coneixement antic amb l'objectiu d'entendre les noves situacions [6]. Això fa que, en termes fonamentals, CBR sigui força diferent a la majoria de les disciplines de la IA[4]. En aquestes, en general, s'utilitza el coneixement expert en el domini d'un problema o es fan associacions generalitzades entre les descripcions i/o conclusions dels problemes. En canvi, el raonament basat en casos fa ús de coneixement concret d'experiències passades, emmagatzemant-lo en *casos*, els quals s'utilitzen, adaptant-los si s'escau, a problemes futurs similars. Per cada problema que es resol, es guarda la seva solució concreta, posant-la a disposició d'altres problemes futurs; d'aquesta manera, es proporciona un aprenentatge prolongat i incremental. [5] [8]

[4] Intel·ligència Artificial

REPRESENTACIÓ I EMMAGATZEMATGE DELS CASOS

Enregistrar un cas significa guardar una descripció del cas a més de la solució que ha sigut capaç solucionar el problema. El conjunt dels casos s'emmagatzema dins de la *case library* o *case base*[5].

Les llibreries de casos, generalment, es divideixen en dos tipus: les memòries lineals i les memòries jeràrquiques, que utilitzen diferents processos de discriminació per a filtrar els casos a consultar [8]. Tenint en compte les seves característiques principals, podem distingir-les de la següent manera:

Memòries lineals	Memòries jeràrquiques
Sempre retornen els casos més semblants al cas actual	Poden deixar de banda casos òptims degut a una mala discriminació
Poc costós d'afegir nous casos	Mantenir la memòria organitzada afegeix un cost més al afegir un nou cas
Temps d'adquisició de casos lent	Temps d'adquisició més eficient

Taula 1. Diferència entre memòries lineals i jeràrquiques

En quant a l'estructura d'un cas, aquest sol contenir característiques com les següents [8]:

- Un identificador únic pel cas.
- Una descripció del cas.
- El diagnòstic del cas.
- La solució acceptada pel cas.
- La correctesa de la solució: si ha sigut satisfactòria o no pel problema.

[5] Llibreria de casos

EL CICLE DE CBR

El procés d'aprenentatge de CBR és bàsic en tota implementació d'aquest. Des d'un punt de vista molt general, aquest cicle es pot resumir en els següents quatre passos (*the 4 REs*) [5] [8]:

1. *Retrieve*. Recuperar el cas o casos més similars a l'actual.
2. *Reuse*. Reutilitzar la informació dels casos antics per solucionar el problema. (Adaptació)
3. *Revise*. Revisar la solució proposada. (Avaluació)
4. *Retain*. Retenir les parts de la solució que poden ser útils per a següents problemes, dins de la llibreria. (Aprenentatge)

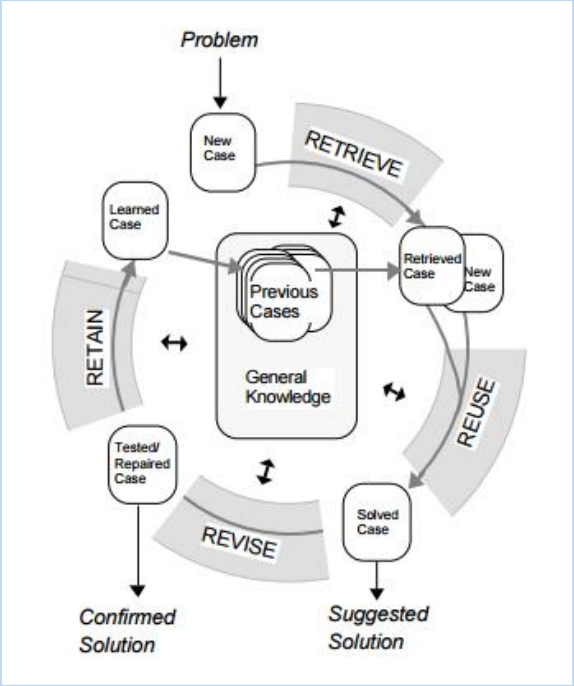

Figura 1. Cicle CBR

Quant a les *Travel Search Engines*, la literatura al respecte és pràcticament inexistent, tant per a les basades en CBR com per a les basades en altres sistemes, principalment perquè els desenvolupadors d'aquestes plataformes pertanyen al sector privat. Conseqüentment, les úniques fonts d'informació que tenim respecte a com funcionen els esmentats sistemes, són les seves pròpies patents (e.g. [2], [3]). Aquesta falta d'informació provoca que no es pugui aprofitar ni adaptar cap solució existent, sinó que haguem de dissenyar una de nova completament.

Tot i així, és evident que la majoria de plataformes utilitzen tècniques properes a CBR de manera directa, emmagatzemant, per cada parella de ciutats, la solució més econòmica en general. Aquesta mecànica es pot veure molt clarament a la pàgina web *Rome2Rio*, on bàsicament se li mostren a l'usuari les combinacions de transports que solen donar millors preus, i, a partir d'aquí, l'usuari decideix quina combinació explorar i buscar els preus pel dia que vulgui.

2.3 CONCLUSIÓ

En una aplicació com la que es pretén en aquest projecte, on la prioritat de treballar amb informació actualitzada respecte a la disponibilitat i el preu dels passatges provocarà que per cada cerca d'un nou usuari haguem de consultar o corroborar a altres webs aquesta informació amb el seu conseqüent (i inevitable) cost temporal (que molt probablement acabi ocupant la majoria del gruix de temps per cada cerca), la idea de treballar amb els algorismes de cerca com els comentats anteriorment sobre un graf amb la informació de tots els viatges disponibles és inassumible, ja que la seva complexitat temporal és massa elevada, tenint en compte que un dels objectius és que la resposta a l'usuari sigui en qüestió de pocs segons.

Tot i això, el que sembla que cap sistema de *Travel Search Engines* ha implementat, és la fusió dels dos conceptes explicats: els algorismes de cerca més CBR. Però, com? La idea que es presenta en aquest projecte és la de mantenir una base de dades en forma de graf, on els nodes representin les ciutats a l'Argentina, i les relacions "viatges" entre aquestes ciutats. Però no

viatges obtinguts en temps real, sinó viatges que es vagin actualitzant a la base de dades amb tècniques de CBR, i que representin el cost mitjà d'arribar de la ciutat d'origen a la de destí. Amb aquest tipus de graf, no tant complex com un mantingut en temps real, podríem aplicar algorismes de cerca sense la por a perdre una gran quantitat de temps explorant-ho, ja que seria molt menys dens.

3. DEFINICIÓ DE L'ABAST

3.1 ABAST

Per aconseguir resoldre el problema que es planteja al projecte, haurem de crear una pàgina web disponible *online* on l'usuari, com a mínim, pugui introduir una ciutat d'origen i una de destí, juntament amb una data de sortida i el número i el tipus de passatgers, clicar al botó de cercar, i que li apareguin en qüestió de pocs segons (uns 10) els primers resultats disponibles; al cap d'uns segons més (fins a 30), s'haurien d'haver mostrat totes les possibilitats que el sistema ha pogut trobar per arribar al destí. Aquestes serien les característiques indispensables que ha de complir la plataforma; altres, com la cerca d'un viatge de tornada, la consulta de preus dins d'un mes sencer, l'opció de poder anar "a qualsevol lloc" (per trobar la destinació més econòmica des de l'origen), seleccionar l'origen/destí com a una província o conjunt de ciutats en comptes de com una ciutat concreta, etc., queden sense especificar i es deixen com a definibles més endavant, en cas de que la funcionalitat bàsica del projecte sigui totalment correcta i es disposi de temps per implementar-les.

Per arribar a crear la pàgina web, caldrà primer realitzar el *back-end*[6]. Aquest, rebent la informació introduïda per l'usuari (que evidentment simularem mentre no hi hagi *front-end*[7]), haurà de consultar la base de dades, obtenir els camins més ràpids per arribar al destí, buscar la informació sobre els trajectes d'interès a webs de tercers (*scraping*), i finalment, actualitzar la base de dades amb informació sobre les noves solucions.

Així, caldrà realitzar un estudi sobre quina és la millor manera d'implementar CBR, és a dir, com aplicar els 4 passos generals del cicle de CBR al projecte, estudiar quin algorisme es pot utilitzar per cercar els camins més econòmics, i trobar una estructura adequada per a la base de dades.

A més, caldrà trobar el llenguatge de programació adequat tant per programar el cicle CBR i la cerca dels camins com per a l'*scraping*, que sigui ràpid i que permeti la paral·lelització de

[6] Capa de dades i càlcul
[7] Capa de presentació/interacció amb l'usuari

tasques, característica important per a implementar *scraping* eficientment. Si és un llenguatge que el desenvolupador desconegui, caldrà que no sigui excessivament complicat d'aprendre.

Finalment, quan el *back-end* respongui bé als jocs de proves, es podrà passar a desenvolupar el *front-end*. Aquest, com s'ha dit, tindrà format de pàgina web. Com que el nostre desenvolupador no és expert en disseny web, no prioritzarem que la pàgina web tingui un disseny excel·lent, sinó que l'objectiu principal és que tingui una funcionalitat correcta (amb les funcionalitats bàsiques especificades abans) i es comuniqui bé amb el *back-end*. Millores al disseny web seran sempre posteriors al correcte funcionament del sistema.

Creiem que les funcionalitats descrites anteriorment són les que hauria de complir el nostre projecte a la finalització d'aquest, al cap de 4 mesos.

3.2 OBSTACLES

A continuació es comenten els possibles obstacles que poden sorgir al projecte i com evitar-los.

3.2.1 ERRORS AL CODI

Els errors al codi són una situació comú quan es desenvolupen projectes grans. Aquests, evidentment, s'han de minimitzar i eliminar el més ràpid possible. Per fer-ho el que es pot realitzar són proves constants de la satisfactibilitat del codi programat a cada moment, i no esperar a tenir programat un gran bloc de codi per a començar a fer proves.

3.2.2 CANVI EN EL FORMAT DE TERCERES WEBS

Encara que seria molt desafortunat dins del termini de finalització del projecte, podria passar que es fes l'*scraping* d'una pàgina web i al cap d'un temps aquesta canviés el seu format, fent que les funcions que extreien informació d'ella quedessin obsoletes. Per això, cal anar revisant periòdicament el funcionament de les funcions i el disseny de les pròpies webs. Independentment, al finalitzar el projecte, s'ha d'anar seguint el mateix procediment constantment per evitar errors al sistema que deixessin als usuaris descontents.

3.2.3 CALENDARI

Al cap i a la fi, el calendari d'aquest projecte és estret, així que no es poden permetre distraccions ja que afectarien seriosament al compliment de la data de finalització.

3.3 METODOLOGIA I RIGOR

Per aconseguir poder tenir el projecte enllestit en el temps establert, s'haurà de seguir una metodologia de treball estricta que asseguri el compliment de les fites al llarg del temps, i que al mateix temps asseguri una qualitat en el treball. A continuació es descriuen les metodologies de treball concretes que es portaran a terme.

3.3.1 TREBALL AMB PROTOTIPS

Per assegurar no arrossegar codi erroni durant molt de temps, és adequat anar generant prototips de les diferents fases de la implementació, per poder provar-los i validar-los conseqüentment; per a cada prototip seria ideal fer un joc de proves mínimament complet per poder assegurar el seu funcionament adequat. Quan estem segurs de que el codi és

completament correcte, el podem guardar com un tipus de *backup*, i continuar cap a la generació del següent prototip. És ideal que es generin prototips constantment i de manera fluïda, per a tenir menys possibilitats de treballar amb grans quantitats de codi erroni.

Aquest, a més de ser un mètode que assegura una validació constant del codi, també ens permetrà mantenir un ritme de treball continu, que ens motivarà més al veure d'una manera "material" que avancem en el nostre projecte.

3.3.2 LES REUNIONS AMB EL DIRECTOR

Les reunions amb el director del projecte són la principal eina de seguiment dins d'aquest, ja que ell és qui fa de suposat "client", i pot anar veient que el projecte, efectivament, avança; a la vegada, són un gran mètode de validació, ja que ell és qui millor criteri té per dir si el projecte va en la bona direcció o no.

3.3.3 ÚS DE GIT

L'ús dels repositoris online *git* serà una manera indispensable d'anar guardant els prototipus del projecte, i de, a la vegada, mantenir-lo endreçat. Aquests repositoris permetran treballar des de qualsevol lloc encara que no es porti l'ordinador a sobre, a més de ser una bona manera de mantenir el projecte guardat a un lloc segur, no susceptible a pèrdues.

4. MODELITZACIÓ DEL DOMINI

A l'hora de calcular la millor combinació de transports per a un dia en concret entre una ciutat i una altra, el mètode ideal seria tenir un DAG[8], on els nodes fossin les ciutats i les arestes els transports entre una ciutat i una altra, que estigués poblat amb tot el conjunt de transports existents entre cada parella de ciutats des del dia de sortida en endavant. Amb un graf com aquest només caldria aplicar algorismes per trobar els *k* camins òptims entre els dos nodes que representin la ciutat de sortida i la d'arribada, trobant així les *k* maneres més econòmiques d'arribar d'una ciutat a una altra, amb les combinacions necessàries.

Una solució com aquesta és totalment inviable principalment per tres motius: el primer i el més obvi, perquè simplement no tenim accés a una base de dades amb tota aquesta informació; el segon, perquè encara que tinguéssim mètodes per a obtenir aquesta informació (de fet, els tenim, ja que només caldria cercar a les pàgines web adequades els transports entre cada parella de ciutats), els preus i la disponibilitat dels transports varien constantment, i hauríem d'actualitzar tota la informació per cada petició d'un nou usuari – i, no cal dir-ho, això seria molt costós temporalment pel tipus d'aplicació que es pretén; i el tercer, perquè simplement seria un graf massa dens per abordar-se mitjançant algorismes de cerca.

Afortunadament, existeixen maneres més eficients d'aproximar-nos a aquesta solució, basades en el paradigma d'intel·ligència artificial CBR, o Cased-Based Reasoning. La majoria de *Travel Search Engines* solen resoldre aquest problema mitjançant la construcció d'una base de dades relacional que emmagatzema una fila per cada ciutat, aquesta amb una columna per cadascuna de les altres ciutats, amb informació sobre les combinacions (ciutats intermèdies, preus mitjans, etc.) que solen ser més econòmiques des de la primera ciutat fins la segona. És una solució que s'ha demostrat factible, però es pot anar més enllà si es pensa en un altre concepte de base de dades, més relacionat amb la naturalesa del nostre problema.

La solució que es proposa pretén aplicar CBR de la següent manera: per cada nova petició de l'usuari, la qual requerirà – obligatòriament, degut a la variabilitat dels preus i la disponibilitat explicada - de cerca (o *scrape*) a les pàgines web abans esmentades, guardar-nos tota la informació obtinguda dins d'una base de dades en forma de DAG. Més tard, aquesta informació guardada per cada petició, servirà per generar noves i millors solucions per altres usuaris. No

[8] *Directed Acyclic Graph*

s'haurà de patir, en un principi, per l'alta connectivitat del graf, ja que en cap cas tindrem un graf tan dens com un omplert amb tots els viatges disponibles.

LES BASES DE DADES ORIENTADES A GRAFS

Abans d'explicar en detall la proposta per solucionar el problema en qüestió, cal definir el tipus de base de dades que s'utilitzarà, ja que, com és evident, és un dels elements fonamentals a CBR. En comptes d'utilitzar un tipus de base de dades relacional convencional, s'optarà per aplicar, com s'ha dit, un model més relacionat amb la naturalesa del problema, una base de dades orientada a grafs (o BDOG).

Aquestes bases de dades, basades en general en *NOSql* (és a dir, en tècniques d'emmagatzematge i obtenció de dades diferents a les utilitzades a les bases de dades relacionals), utilitzen estructures de grafs amb nodes i relacions – unidireccionals o bidireccionals -, amb les seves corresponents propietats. Les BDOG van ser dissenyades per a permetre un *retrieval* simple i, a la vegada ràpid, dins d'estructures jeràrquiques complexes, on les bases de dades relacionals tradicionals, utilitzarien, per aconseguir els mateixos resultats, peticions molt més complexes i, en general, amb molta menys eficiència. Les BDOG estan experimentant un creixement considerable als últims anys degut a la facilitat que aporten per treballar amb les xarxes socials i el *big data* en general.

A continuació s'explica més en profunditat com és i quines característiques té aquesta proposta.

4.1 PROPOSTA DE SOLUCIÓ

En concret, per resoldre el problema, s'utilitzarà una BDOG on s'emmagatzemin: nodes, que representen ciutats a l'Argentina; *relacions de tipus General*, que representen el preu mitjà per arribar d'una ciutat a una altra amb un transport determinat (doncs, amb les propietats de Preu i Tipus de Transport[9]); i *relacions de tipus Específic*, que representen viatges concrets entre una ciutat i una altra (doncs, amb les propietats de Preu, Tipus de Transport, Data de Sortida i Data d'Arribada). La justificació de la utilització d'aquests dos tipus de relacions en comptes d'un de sol (les Generals) es basa en que, tot i que les relacions Generals ens aporten informació sobre com sol costar arribar d'una ciutat a una altra, les relacions Específiques ens aporten informació d'un dia i hora en concret, que pot ser molt més útil degut a la casualitat dels preus dels viatges. Per exemple, si volguéssim trobar les combinacions entre A i B a la Figura 2 i només tinguéssim en compte les relacions generals (és a dir, el preu mitjà dels viatges), obtindríem només la combinació A-D-B. En canvi, si fem una cerca tenint en compte les relacions específiques (és a dir, el preu de viatges concrets que es coneixen pel dia en qüestió), trobaríem que específicament aquell dia és més econòmic arribar a B amb la combinació A-C-B. Això, en cap cas vol dir que la solució A-C-B sigui més econòmica que la solució A-D-B, ja que, o bé els preus dels viatges "reals" d'A-C-B poden haver pujat (o desaparegut), o bé han aparegut viatges més econòmics a A-D-B.

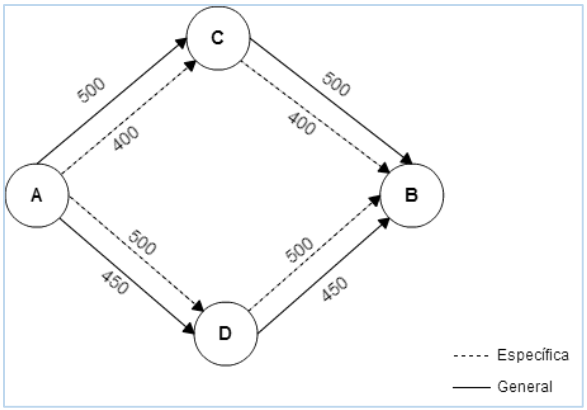

Figura 2. Exemple de trajectes entre A i B. S'obvien propietats com el tipus de transport i les dates.

[9] Quan parlem de "Tipus de Transport" ens referim bàsicament a tres tipus, que representen els tres webs d'on obtenim la informació: Aerolíneas Argentinas (www.aerolineas.com.ar, per avió), LAN Airlines (www.lan.com, per avió) i Plataforma 10 (www.plataforma10.com.ar, per autobús).

D'ara en endavant, i per evitar confusions, quan parlem de "trajectes" o "viatges directes" ens referirem als viatges entre una ciutat i una altra utilitzant només un transport, és a dir, sense cap combinació. De la mateixa manera, per referir-nos als viatges utilitzant més d'un transport, parlarem de "viatges amb combinacions" o "camins". A més a més, com hem fet abans, quan parlem de trajectes "reals" ens estarem referint als trajectes que es poden trobar a les pàgines web d'on s'extreu la informació.

Un cop definida l'estructura de dades utilitzada sobre on aplicarem CBR, cal explicar com s'aplicarà exactament el paradigma a la nostra solució. La idea és la següent: per cada nova petició d'un usuari, es cercaran dins de la base de dades els *k* camins òptims entre la ciutat d'origen i de destí, primer tenint en compte només les relacions Generals, i després només les relacions Específiques. Aquests dos tipus de relacions es crearan o actualitzaran de manera progressiva a mesura que arribin més peticions al sistema, a partir de la informació que s'obtingui de les webs externes quan s'hagin de corroborar les dades. Per veureu més clarament, a continuació s'explica amb detall l'aplicació del cicle de CBR (les etapes *Retrieve*, *Reuse*, *Revise* i *Retain*) pel que fa el nostre plantejament.

- *Recuperació.* Aquesta etapa fa referència a la obtenció de solucions anteriors de la base de dades. En el nostre cas, una solució és un camí entre la ciutat d'origen i la de destí, i el que volem fer és obtenir, de tots els camins disponibles, els *k* òptims, tant per a les relacions Generals com per les relacions Específiques. Per a definir el procés, cal dividir l'etapa en dues, ja que presenten diferències importants: la *Recuperació* de solucions Generals – només amb relacions generals -, i la *Recuperació* de solucions Específiques – només amb relacions Específiques. Ambdues subetapes utilitzaran l'algorisme K* [9] pel càlcul dels camins òptims, explicat més endavant a l'apartat d'Implementació.

    - *Recuperació* de solucions Generals. Aquesta subetapa no té gaire complexitat, ja que només s'ha de demanar a l'algorisme que calculi els *k* camins òptims entre el node d'origen i el node de destí, utilitzant les relacions Generals tal i com són explícitament al graf. Això retornarà un conjunt de camins que prenem com a conjunt de solucions recuperades.

- *Recuperació* de solucions Específiques. Si bé a primera vista el funcionament és el mateix que al cas anterior, aquest té una particularitat que el fa molt especial, i és que, si es té en compte que les relacions Específiques tenen una data de sortida i una d'arribada, l'elecció d'una aresta d'un node cap a un altre limita el conjunt d'arestes que surten del segon node que es podran escollir. Si prenem d'exemple la Figura 3, veiem que si arribem a la ciutat B des de la relació 1, des de B podrem escollir les dues relacions que surten cap a C (3 i 4); en canvi, si escollim la relació 2, només podrem escollir des de B l'aresta que surt a les 17:00 (4), ja que l'altra (3) ha sortit abans.

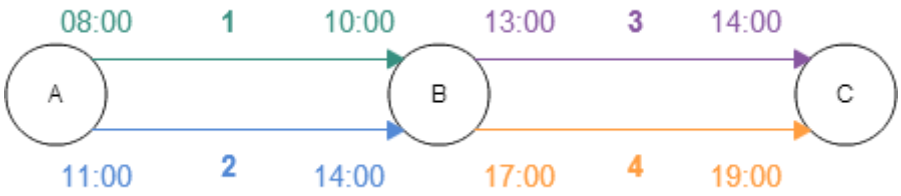

Figura 3. Graf amb relacions Específiques. S'obvien propietats com el tipus de transport i el dia. 1,2,3,4 representen els identificadors de les relacions.

La solució que s'ha ideat per a solucionar aquest problema és *transformar el graf original en un segon graf alternatiu*, on cada node representi una relació Específica del graf original, i on, hi haurà una aresta entre dos nodes si aquestes dues relacions són compatibles, és a dir, si l'hora d'arribada de la relació del primer node més el temps pel transbord és inferior o igual a l'hora de sortida de la relació del segon node i no superior a les hores màximes d'espera per transbordar; el pes d'aquesta aresta serà igual al preu de la relació del segon node. A més, hi afegim un node *a* que tindrà una aresta cap a cada node que representi una relació original que surti de la ciutat de sortida (amb pes igual al preu de la relació corresponent), i un node *b* al qual arribarà una aresta per cada node que representi una relació original que arribi a la ciutat d'arribada (amb pes 0). A mode d'exemple, veiem a la Figura 4 com es transformaria el graf de la Figura 3 en aquest segon tipus de graf.

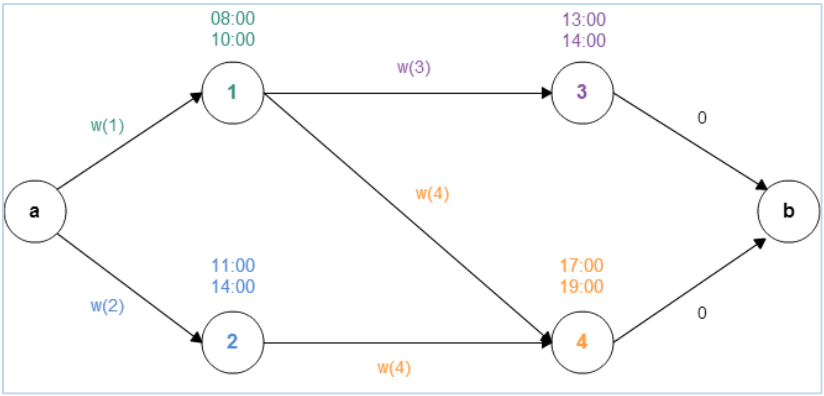

Figura 4. Graf de la Figura 3 transformat. w(i) representa el preu de la relació i al graf original.

El nombre de nodes d'aquest segon graf és, com a màxim, el nombre de relacions de l'original, i el nombre de relacions és, com a màxim, el quadrat d'aquestes. La complexitat que això implicarà a l'algorisme, s'explicarà juntament amb aquest, a l'apartat 6. Un cop tinguem aquest segon tipus de graf podem aplicar-li l'algorisme K* sense tenir por de no complir amb les restriccions de les relacions, i obtenir així el conjunt de camins que prenem com el conjunt de solucions recuperades.

- *Adaptació*. En aquesta etapa s'han d'adaptar les solucions trobades al nou problema. En el nostre cas, el que ens interessa és veure que aquestes solucions són consistents amb la "realitat". És a dir, hem de comprovar, per cada camí, quins trajectes reals hi ha per cadascun dels trajectes dins d'aquest, i si aquests trajectes reals són compatibles entre ells. Dit d'una altra manera, per cada solució – tant General com Específica - , hem de buscar els trajectes trobats a la base de dades a la web externa que correspongui, i, donat que per cada trajecte poden aparèixer varis trajectes reals, hem de seleccionar, d'entre totes les combinacions possibles (les que compleixin les restriccions de temps de transbord abans explicades per les solucions Específiques), les millors segons alguns criteris (preu, temps, ...). Els criteris i l'algorisme per seleccionar les combinacions s'explicaran a l'apartat 6.

- *Avaluació*. Si bé aquesta etapa pot ser més o menys comú a les aplicacions de CBR, en el nostre cas no requerim de cap avaluació de les solucions trobades, ja que són perfectament vàlides per mostrar-se a l'usuari.

- *Aprenentatge*. Aquesta és potser l'etapa més important al procés. És on s'ha de guardar la informació sobre les solucions trobades a la base de dades per a que sigui accessible a posteriors problemes. A més, és on el nostre sistema divergeix de la resta: la majoria de sistemes el que fan és guardar a una base de dades relacional tot o part del conjunt de solucions trobades; en el cas de les *Travel Search Engines*, actualitzant les combinacions "comuns" entre la ciutat d'origen i de destí a partir de les combinacions finals mostrades a l'usuari. En el nostre cas, el que fem és, en comptes de guardar la informació global sobre les solucions trobades, guardar, per cada cerca a les webs externes a l'etapa d'*Adaptació* amb l'objectiu de trobar els trajectes reals d'una ciutat A a una altra ciutat B i amb un transport T, la informació sobre aquests trajectes reals a la nostra base de dades, de la següent manera: actualitzant el preu mitjà de la relació General entre A i B pel transport T (si existeix, sinó es crea una nova), i creant una nova relació Específica entre A i B amb transport T, amb el preu i l'hora de sortida i d'arribada del trajecte real en qüestió (si encara no existeix).

La solució proposada, té l'avantatge en front a la resta de sistemes de Travel Search Engines, que, en comptes de veure les solucions als problemes com a entitats individuals sense cap tipus de comunicació entre elles (saber el preu mitjà de les combinacions més econòmiques entre una ciutat i una altra no ajuda a millorar el coneixement que es té entre una altra parella de ciutats), veu les solucions als problemes com a un conjunt de sub-solucions – els trajectes - , que si bé poden servir per millorar en un futur el mateix problema, també poden millorar qualsevol altra problema, si algun dels seus trajectes millora un camí entre una altra parella de nodes.

5. DISSENY DE L'APLICACIÓ

5.1 ANÀLISI DE REQUERIMENTS

A l'anàlisi de requeriments s'especificarà el que s'espera que ofereixi el nostre sistema als usuaris i de quina manera. Primer es farà de forma textual amb els requeriments d'usuari, i després de manera gràfica amb el diagrama de casos d'ús.

5.1.1 REQUERIMENTS D'USUARI

Per entendre correctament quines operacions s'espera que el nostre sistema realitzi a l'hora d'interactuar amb l'usuari, com també quines qualitats i restriccions ha de tenir aquesta interacció a la plataforma web, ens cal definir quins són els requeriments funcionals i els requeriments no funcionals d'usuari.

REQUERIMENTS FUNCIONALS

A continuació es descriuen els diferents requeriments funcionals d'usuari. S'especifiquen les funcions que s'espera que compleixi el sistema davant de les accions dels usuaris, amb l'objectiu d'entendre millor quines funcionalitats podrà realitzar exactament el nostre sistema.

Les funcions s'escriuran en l'ordre en que s'executarien al sistema, seguint el flux d'actuació que tindria un usuari qualsevol.

A. L'usuari serà capaç de connectar-se i utilitzar al sistema a través de la web.

Quan un usuari es connecti a la pàgina web, el servidor del sistema ha de respondre amb el codi i els elements necessaris per generar aquesta web al navegador de l'usuari. La interfície que es mostrarà a l'inici és la del formulari principal per introduir les dades del viatge.

B. L'usuari podrà enviar la informació referent a la seva cerca.

Un cop la interfície ha sigut carregada, l'usuari ha de ser capaç d'omplir el formulari corresponent i d'enviar les dades al servidor. Entre les dades que l'usuari hauria d'enviar, s'haurien d'incloure obligatòriament: la ciutat de sortida i d'arribada, el dia, mes i any de sortida, i el número d'adults i de nens. Altres dades dependran de si s'arriben a implementar o no funcionalitats addicionals.

C. El sistema redirigirà a l'usuari a una nova interfície.

Al rebre la petició de l'usuari, el sistema enviarà a l'usuari a una nova pàgina al seu navegador on més tard apareixeran les solucions que es trobin.

D. El sistema buscarà les solucions.

Un cop rebuda la petició de l'usuari demanant les solucions, el sistema executarà les funcions corresponents per buscar tant els viatges directes com els viatges amb combinacions, a partir de la informació introduïda per l'usuari.

E. El sistema mostrarà les solucions a l'usuari.

Un cop el sistema hagi calculat i obtingut totes les solucions, les mostrarà a l'usuari omplint l'espai corresponent a la nova interfície a la qual se li ha redirigit. Per cada solució es mostrarà: el dia i l'hora de sortida, el dia i l'hora d'arribada, el preu total, el preu per adult i el número i el tipus (avió o bus) de trajectes. Per cada trajecte, a més, es mostrarà el nom de la companyia, el lloc de sortida i d'arribada (aeroport / estació d'autobús, amb la ciutat), l'hora de sortida i d'arribada, el preu total, el preu per adult, i un enllaç a la pàgina web corresponent per comprar el bitllet.

F. L'usuari podrà filtrar les solucions.

La interfície proporcionarà eines a l'usuari per a que pugui filtrar les solucions en funció de paràmetres com: el preu màxim, l'hora de sortida mínima i màxima, el dia d'arribada, el nombre d'escales màximes, etc.

G. L'usuari podrà ordenar les solucions.

La interfície proporcionarà eines a l'usuari per a que pugui ordenar les solucions en funció d'algun dels següents criteris: preu total (per defecte), preu per adult, hora de sortida, hora d'arribada i nombre d'escales.

H. L'usuari serà capaç de canviar la cerca sense tornar a la pantalla d'inici.

La interfície de mostra de solucions permetrà, a més, que l'usuari torni a generar una nova cerca, tornant a introduir els paràmetres del formulari i reenviant-lo. A més, l'usuari podrà canviar la data de sortida a demà o ahir sense necessitat de tornar a omplir tot el formulari, per mitjà d'un botó específic.

I. El sistema tornarà a executar el procés de cerca sense necessitat de recarregar la pàgina.

Quan l'usuari decideixi canviar la cerca des de la pantalla de mostra de solucions, no caldrà recarregar la pàgina sencera per mostrar les noves solucions trobades pel sistema, sinó que només s'actualitzarà la llista de solucions.

REQUERIMENTS NO FUNCIONALS

Els requeriments no funcionals d'usuari especificaran quines característiques han de complir els requeriments funcionals especificats anteriorment.

A. Temps de resposta.

Un cop l'usuari ha introduït i enviat la informació del formulari, el sistema hauria de respondre amb les solucions directes (si n'hi ha) en un temps de menys de 10 segons, i amb les solucions amb combinacions en un temps de menys de 30 segons.

B. Usabilitat.

La interfície web del sistema ha de ser fàcilment utilitzable i comprensible, amb una corba d'aprenentatge molt ràpida per a qualsevol tipus d'usuari.

C. Disponibilitat.

El servidor web ha de estar sempre disponible per ser utilitzat pels usuaris.

D. Robustesa (*front-end*).

Si l'usuari introdueix algun error a l'hora d'omplir els formularis de la interfície, aquests s'han de notificar a l'usuari de la manera més atractiva possible i sense destruir el flux d'execució.

E. Consistència de les solucions.

Les solucions que es mostrin a l'usuari (els viatges, tant directes com amb combinacions), han de tenir un estat consistent. És a dir, han d'equivaldre a trajectes reals que es puguin comprar a les respectives pàgines webs i han de tenir les mateixes característiques (preu, horari, ...).

5.1.2 DIAGRAMA DE CASOS D'ÚS

Al següent diagrama de casos d'ús (Figura 5) il·lustrem el que s'ha explicat als requeriments funcionals, amb totes les accions que hauria de poder prendre l'usuari un cop s'hagi connectat a la pàgina web.

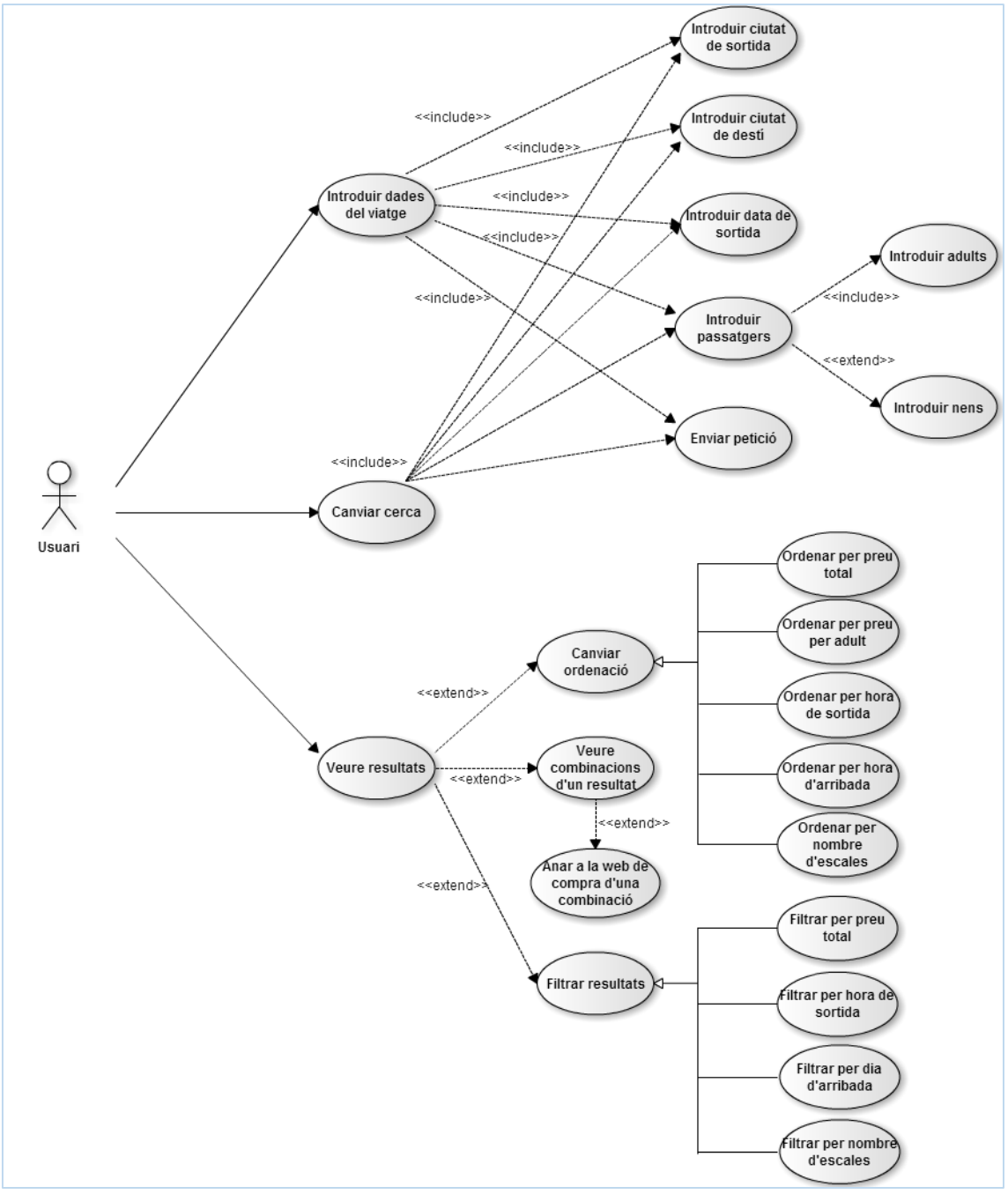

Figura 5. Diagrama de casos d'ús del sistema.

5.2 ESPECIFICACIÓ

En aquest apartat es defineixen les característiques i l'organització del *back-end*. Primer s'expliquen quins requeriments no funcionals (característiques o restriccions) tindrà el nostre sistema, després es comenta la modelització de les dades amb la divisió en classes del domini, i finalment es descriu el flux d'execució i la seqüència d'operacions dins del sistema.

5.2.1 REQUERIMENTS NO FUNCIONALS DEL SISTEMA

Els següents són els requeriments no funcionals que haurà de complir el nostre sistema, alguns més presents a llarg termini que a curt termini, i d'altres presents des del començament del projecte.

A. Aprenentatge.

El sistema ha d'anar millorant les solucions proposades als usuaris (és a dir, trobant solucions més econòmiques) a mesura que vagi omplint i actualitzant el graf (aprenent) a partir de les peticions dels mateixos.

B. Dependència de tercers.

El sistema dependrà de l'estat en el que es trobin les webs d'on s'extregui la informació dels trajectes. Per aquest motiu, s'haurà de revisar si hi ha algun error a l'hora d'obtenir la informació, ja que les webs poden canviar el seu format en qualsevol moment (relacionat amb la funcionalitat **H**). A més, l'estat d'aquests servidors pot influenciar, i força, el temps de resposta del sistema.

C. Manteniment.

El sistema ha de ser revisat periòdicament per a comprovar que no existeixen errors, i, a la vegada, per a possiblement millorar o afegir alguna funcionalitat.

D. Eficiència.

El *back-end* hauria de ser el més eficient possible en quant a utilització de recursos, ja que potencialment poden haver-hi molts usuaris connectats, i no s'hauria de saturar el servidor.

E. Paral·lelització de consultes.

Les consultes a webs externes per a comprovar els trajectes de les solucions seran temporalment el més costós dels nostres algorismes, així que s'haurien d'intentar d'executar totes en paral·lel.

F. Robustesa (*back-end*).

Quan apareguin errors al sistema, aquests no han d'afectar negativament a la navegabilitat de l'usuari i en cap cas han de fer que el servidor s'aturi.

G. Re-usabilitat.

El codi ha de ser altament re-usable, tant amb la divisió de les tasques pròpies en mòduls ben diferenciats com amb la utilització de codi obert de llibreries de tercers.

H. Escalabilitat.

Òbviament, el sistema pot arribar a tenir un graf molt dens a la base de dades. Els algorismes que s'utilitzin, i el sistema en general, ha d'escalar bé amb l'augment de la quantitat de relacions al graf.

5.2.2 MODEL DE LES DADES

Degut a que s'ha utilitzat el paradigma de programació orientat a objectes, les estructures de dades bàsiques es modelitzaran amb un conjunt de classes, que formen la capa del domini dins de l'arquitectura MVC (Model-Vista-Controlador, apartat 5.3.1). Seguidament s'especifiquen les classes del domini a un diagrama UML (Figura 6), tenint en compte els seus atributs i els seus mètodes públics, i dividits en els paquets corresponents. Com totes les classes (excepte els *Handlers*) tenen a veure amb l'algorisme K* de cerca de camins òptims, es deixa l'explicació d'aquestes per l'apartat 6, on s'explicarà l'algorisme.

Hi ha algunes qüestions sobre el diagrama que caldria puntualitzar. Degut a les característiques del llenguatge de programació utilitzat (*Golang*, veure l'apartat d'Implementació), el qual combina funcionalitats de l'*scripting* amb l'orientació a objectes, hi ha entitats del sistema, com ara funcions o *structs*, els quals no es van acabar definint dins de cap classe, sinó a dintre de paquets que els altres s'encarreguen d'importar, pel simple fet que el llenguatge proporciona aquesta comoditat d'estalviar classes a vegades innecessàries. Concretament, parlem de les classes especificades al diagrama com a "Common" i "Scraping", dins dels paquets amb el mateix nom. A l'hora de la implementació, aquestes dues classes s'han implementat com a simplement dos paquets, amb tot un conjunt de *structs* i funcions dintre seu, que són importats pels paquets que els necessitin; ambdós s'han acabat especificant com a dues classes *singleton* per coherència amb la resta de l'especificació. De la mateixa manera, les classes dins del paquet "Common", "Trip" i "Query", són en realitat dos *structs* als quals tots els altres paquets tenen accés; igualment, per coherència s'han definit com a dues classes. Altres funcions o *structs* – secundaris -, definits dintre de paquets en comptes de classes, directament no s'han inclòs al diagrama, i per no portar a confusions no es farà referència explícita a ells als diagrames de seqüència del següent apartat. Finalment, hi ha interfícies al diagrama (concretament, *HTTPHandler* i *Heap*) que són importades d'altres paquets de *Golang*, però que s'inclouen, també, per coherència amb les altres classes que implementen els seus mètodes.

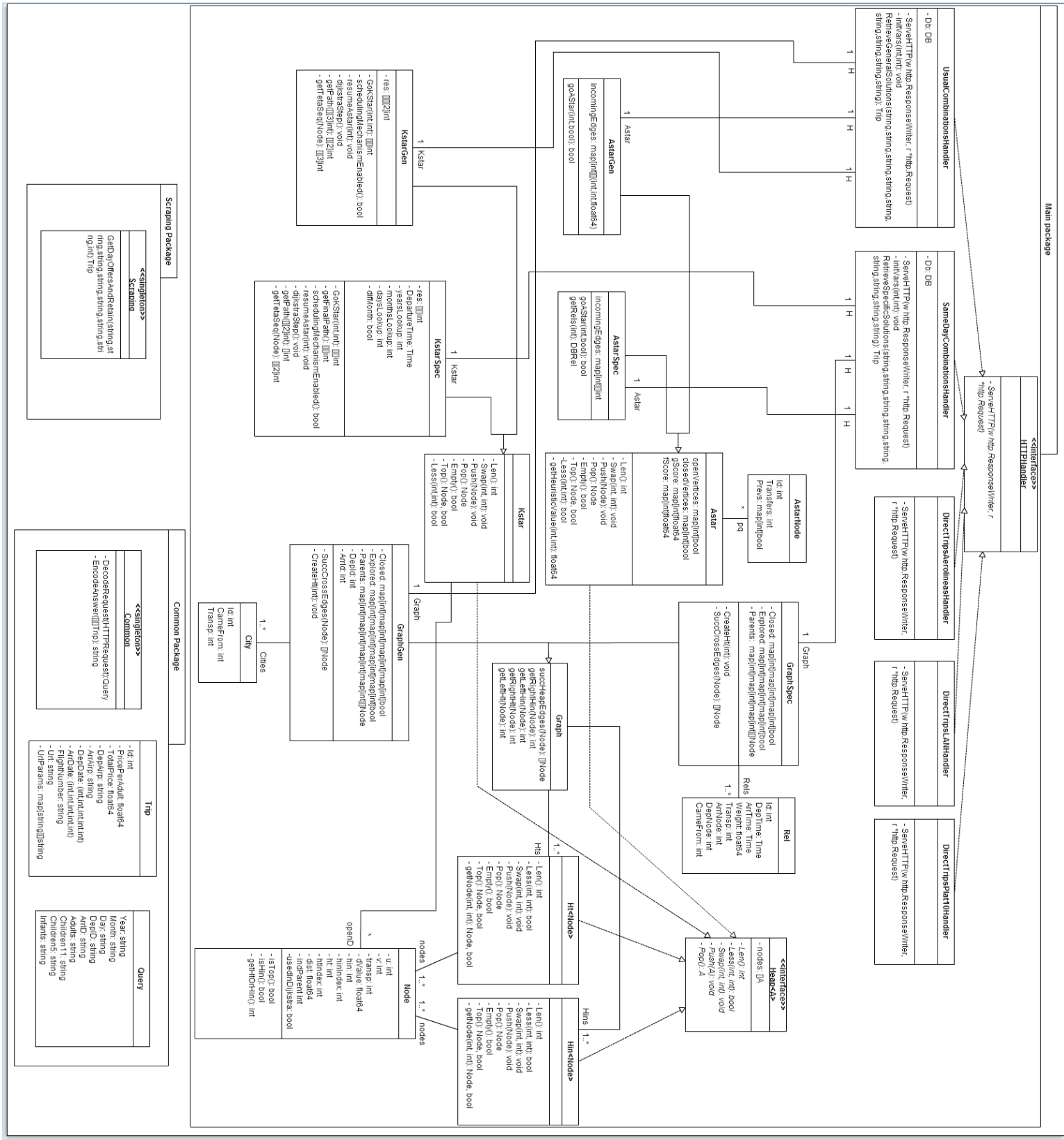

Figura 6. Diagrama de classes UML del domini.

5.2.3 MODEL DEL FLUX DE DADES

En aquest apartat s'introdueix la seqüència operativa del sistema, des de que l'usuari realitza una petició de cerca de viatges fins que se li retornen els resultats. Primer de tot es defineix, des d'un nivell molt alt, l'esquema general d'execució amb un diagrama de flux – que ja es va entreveure a l'apartat 4 -; després, s'entrarà més en detall en les accions que pren cada classe especificada al diagrama de classes del domini amb la construcció d'un diagrama de seqüència UML.

D'una manera molt general, el flux d'execució del sistema pot definir-se de la següent manera (Figura 7).

1. Primer de tot, **l'usuari realitza una cerca amb les dades corresponents**, captada per la interfície. Aquesta cerca provoca que la mateixa interfície demani al sistema tres peticions diferents: obtenir les combinacions amb relacions Específiques, obtenir les combinacions amb relacions Generals i obtenir els viatges directes – aquesta, en realitat, es divideix en tres peticions, una per a cada tipus de transport, però per senzillesa l'expressem com una de sola.
2. Seguidament, **el controlador rep les tres peticions**, que transfereix al *back-end* esperant que aquest respongui amb la informació calculada. Depenent de la informació que s'hagi de calcular es cridaran a unes funcions o a unes altres d'aquest.
3. La **petició arriba llavors al *back-end* en forma de tres crides diferents**. Pel càlcul dels viatges directes, simplement crida a les funcions adequades d'*scraping* per a que retornin els viatges que trobin. Pel càlcul dels viatges amb combinacions, primer crida a les funcions de K* corresponents – per les relacions Generals i per les relacions Específiques – per obtenir els *k* camins òptims, i després passa els camins trobats a un estat consistent comprovant els trajectes de cada camí a les pàgines web corresponents mitjançant els *scrapers*. Els dos tipus de crida als *scrapers*, a la vegada, quan obtinguin la informació de les pàgines web, cridaran a les funcions corresponents per actualitzar les relacions de la base de dades (etapa d'*aprenentatge*).
4. Quan el *back-end* acabi alguna de les funcions, **retornarà el resultat al controlador**.
5. Finalment, el controlador respondrà amb el resultat de la funció a la interfície, que **afegirà els resultats a la llista de solucions trobades**.

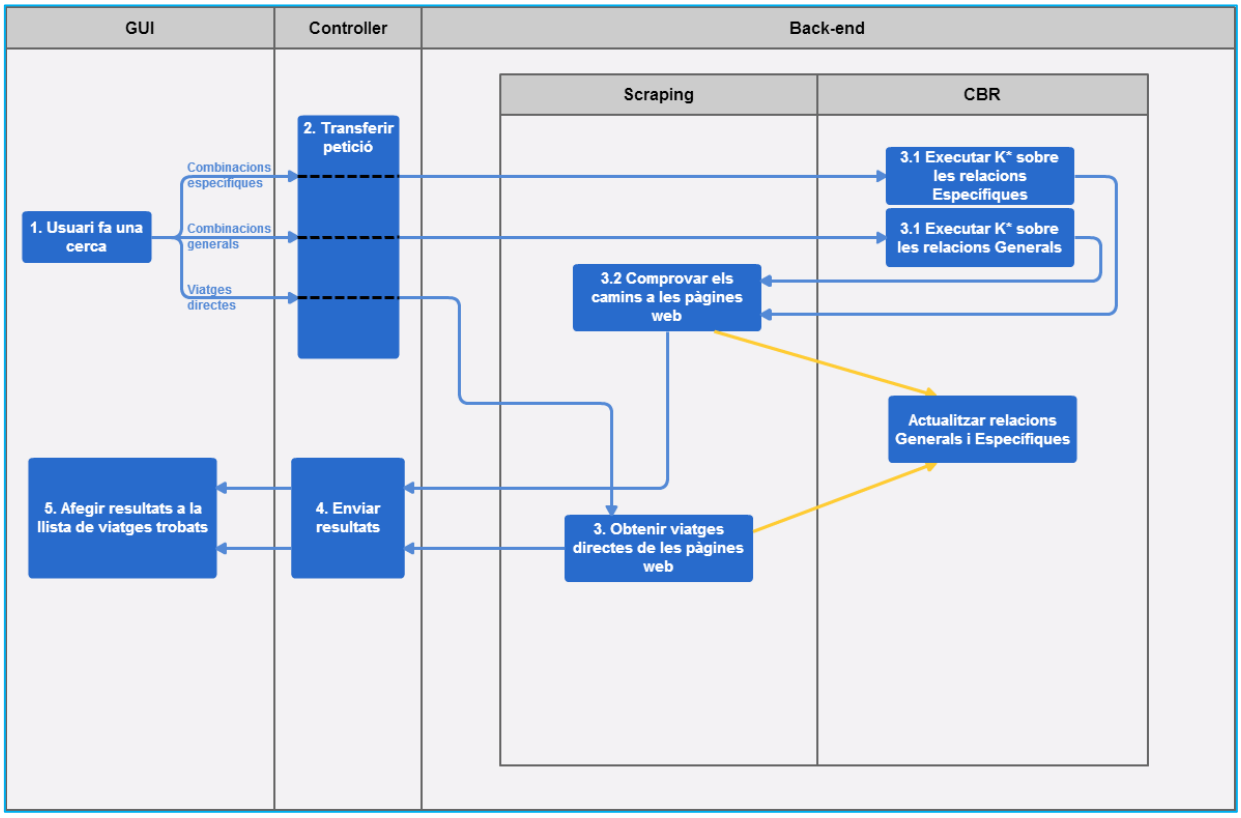

Figura 7. Diagrama de flux del sistema.

Un cop vist el flux general d'operacions del sistema, es definiran, també sense entrar massa en detall en com s'executen totes les funcions i obviant la majoria del codi (es deixa aquesta tasca per l'apartat 6), com es divideix aquest flux entre les classes del domini definides a l'apartat anterior. Es defineix, per a cada *Handler*, la seqüència d'operacions des de que la petició arriba al sistema des de la interfície fins que es retorna a aquesta el conjunt de solucions trobades.

Les figures 8, 9 i 10 mostren els diagrames de seqüència UML de les peticions per obtenir els viatges directes, pels tres tipus de transport (corresponents a les webs Aerolíneas Argentinas, LAN i Plataforma10). Veiem com, per totes tres, bàsicament es crida a la funció per obtenir els viatges directes *GetDayOffersAndRetain*, que retorna tot el conjunt de trajectes per un dia i transport determinat, i realitza l'etapa d'*aprenentatge*, actualitzant la base de dades amb la nova informació d'aquests trajectes.

Les Figures 11 i 12 mostren els diagrames de seqüència UML de les peticions pels viatges amb combinacions de relacions Generals i de relacions Específiques. En aquest cas, es crida a les funcions *RetrieveSpecificSolutions* o *RetrieveGeneralSolutions*, segons el cas. Aquestes, criden a la funció *GoKStar* dels seus respectius atributs *Kstar* (que, com s'ha dit, de moment veiem com a una caixa negra i el seu funcionament s'explicarà més endavant), per obtenir el conjunt de camins entre la ciutat de sortida i la d'arribada, en forma de llista de relacions. Aquest conjunt de camins s'itera, i, per cada camí es troba un conjunt de solucions "consistents" que s'afegeix a la llista que es pot retornar a l'usuari; això es farà iterant cada relació del camí, cridant als respectius *scrapers* per cada una d'elles, i del conjunt d'ofertes obtingudes per cada relació, trobar la manera més adequada de combinar-les (l'algorisme exacte es comentarà a l'apartat 6). Noti's que a les relacions Específiques es crida als *scrapers*, evidentment, amb el dia de sortida que tingui la relació, i en canvi, a les relacions Generals, es crida als *scrapers* pel dia de sortida i pel dia següent, per tenir un rang més o menys ampli de possibilitats.

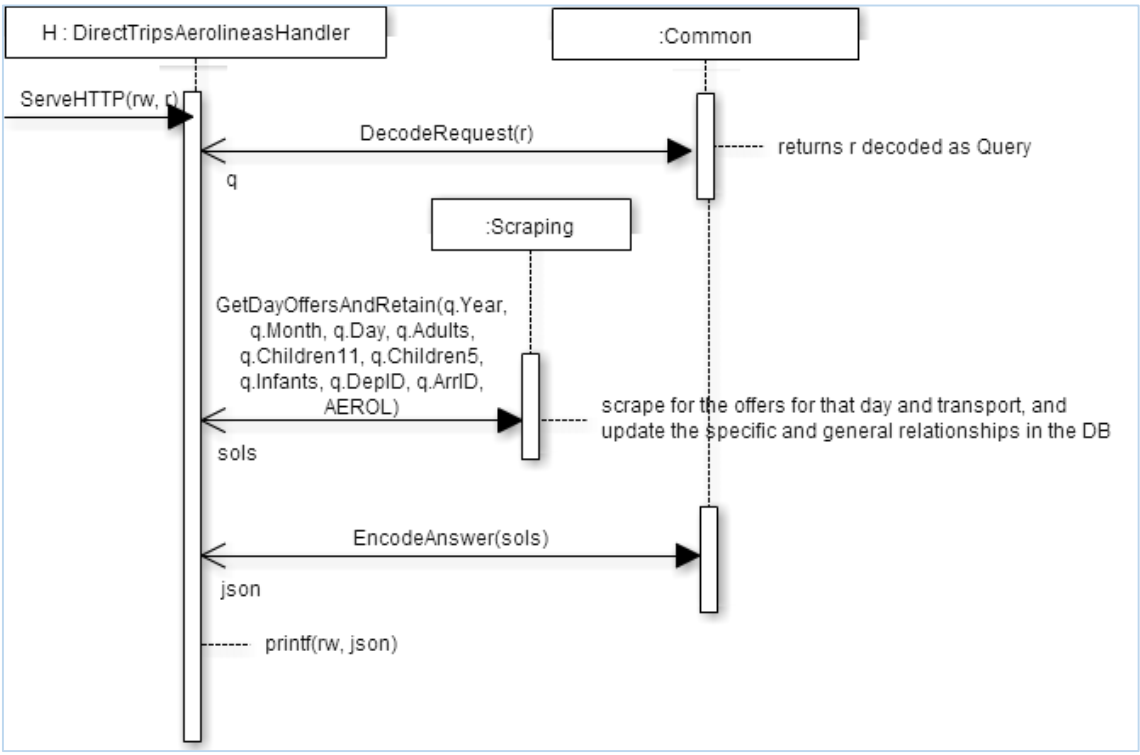

Figura 8. Diagrama de seqüència *DirectTripsAerolineas*.

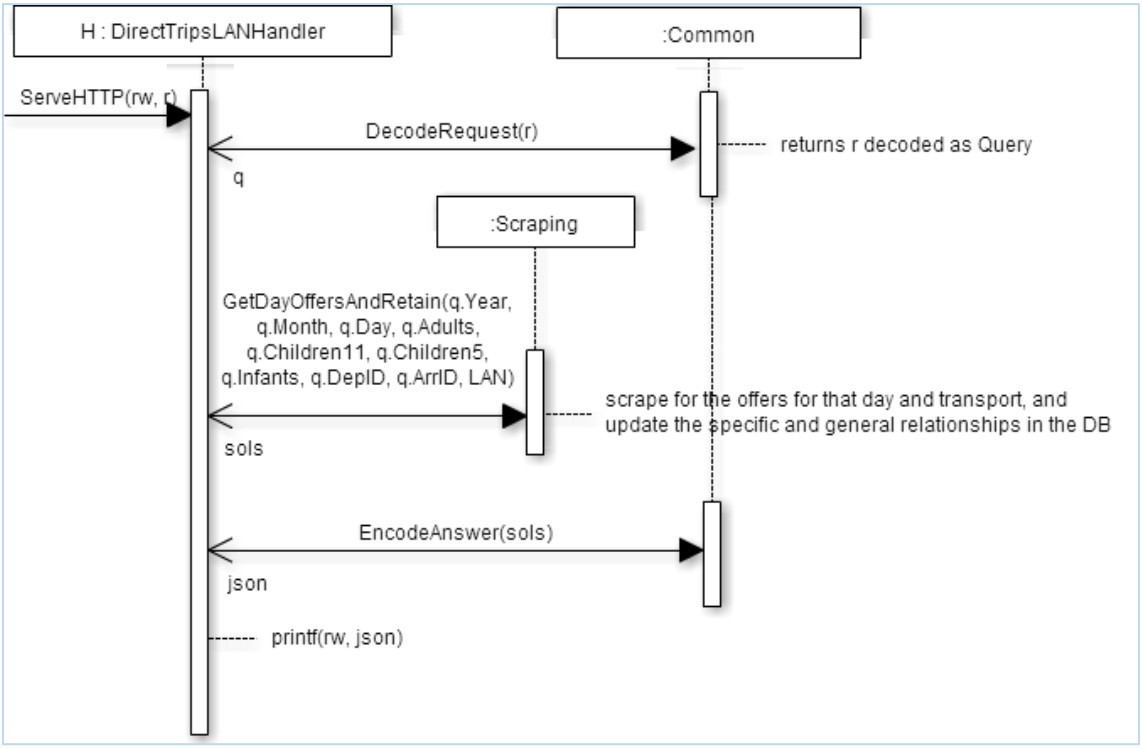

Figura 9. Diagrama de seqüència *DirectTripsLAN*.

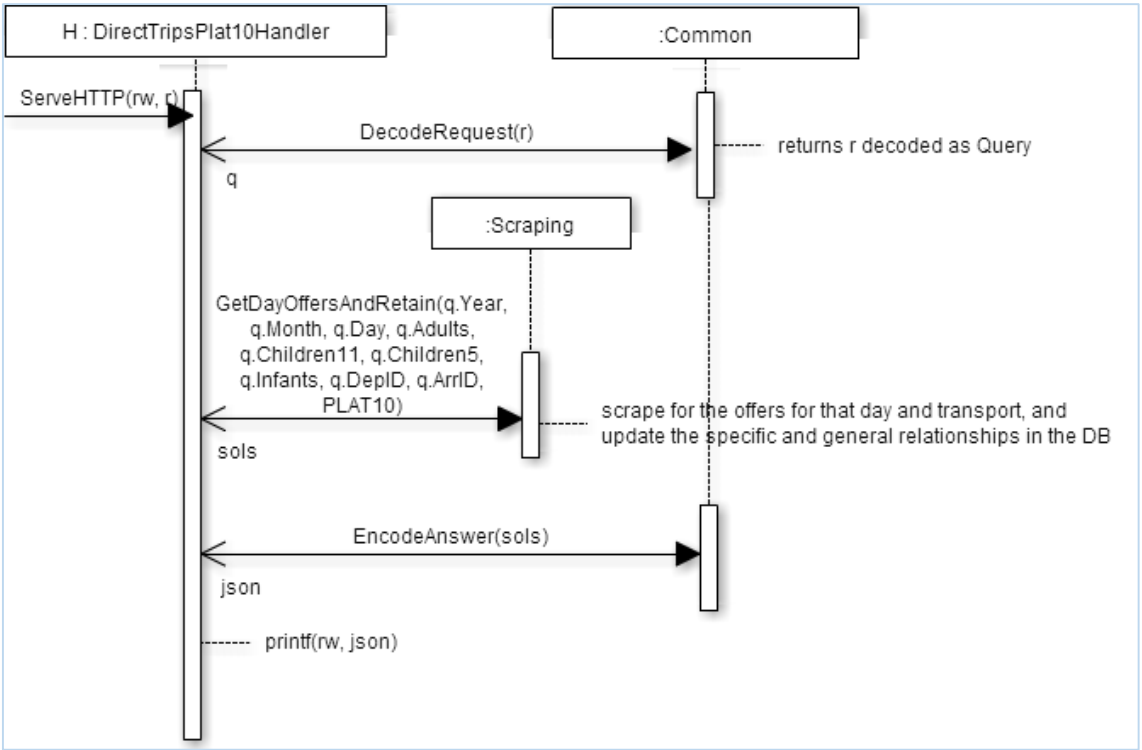

Figura 10. Diagrama de seqüència *DirectTripsPlat10*.

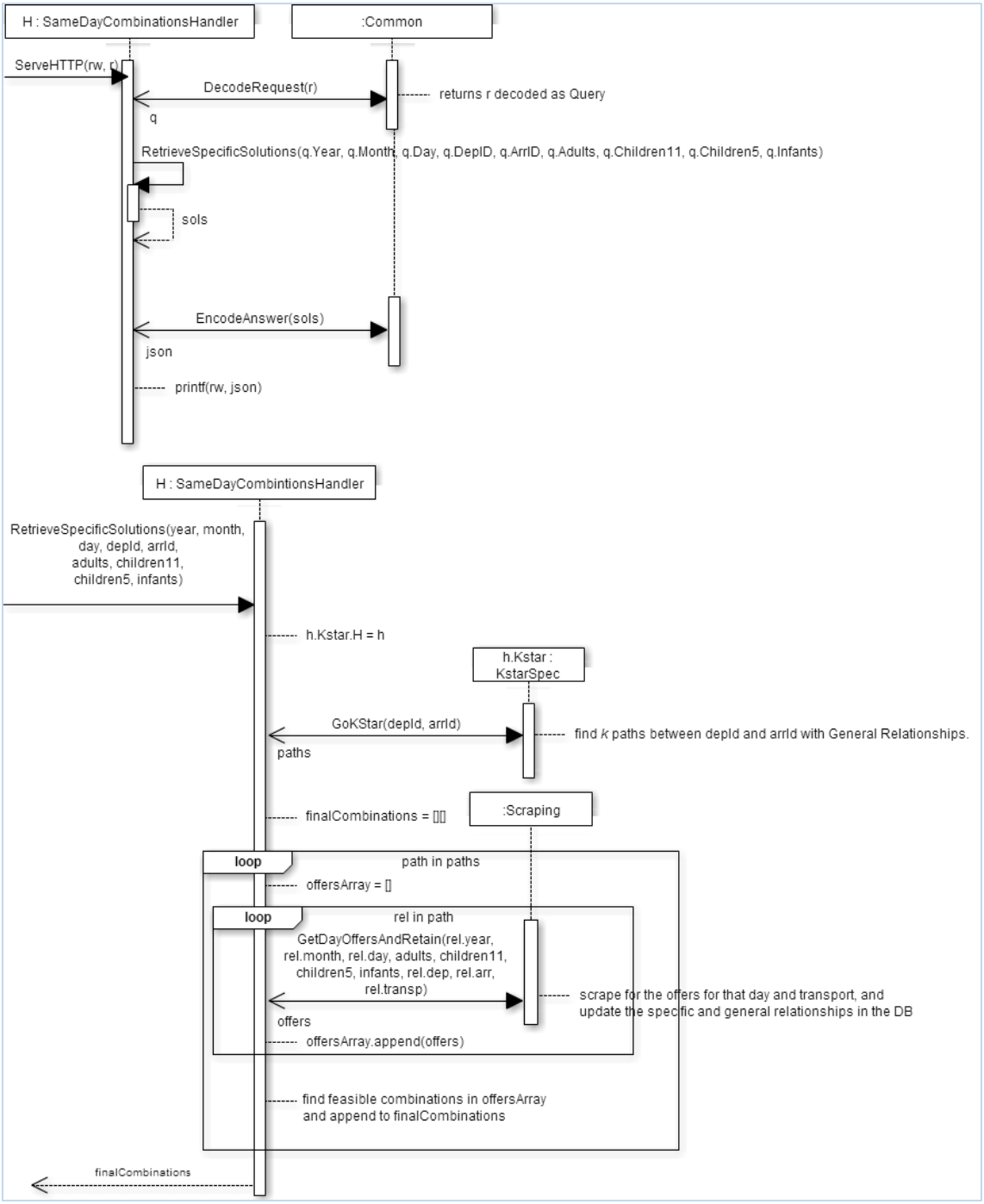

Figura 11. Diagrama de seqüència *SameDayCombinations*.

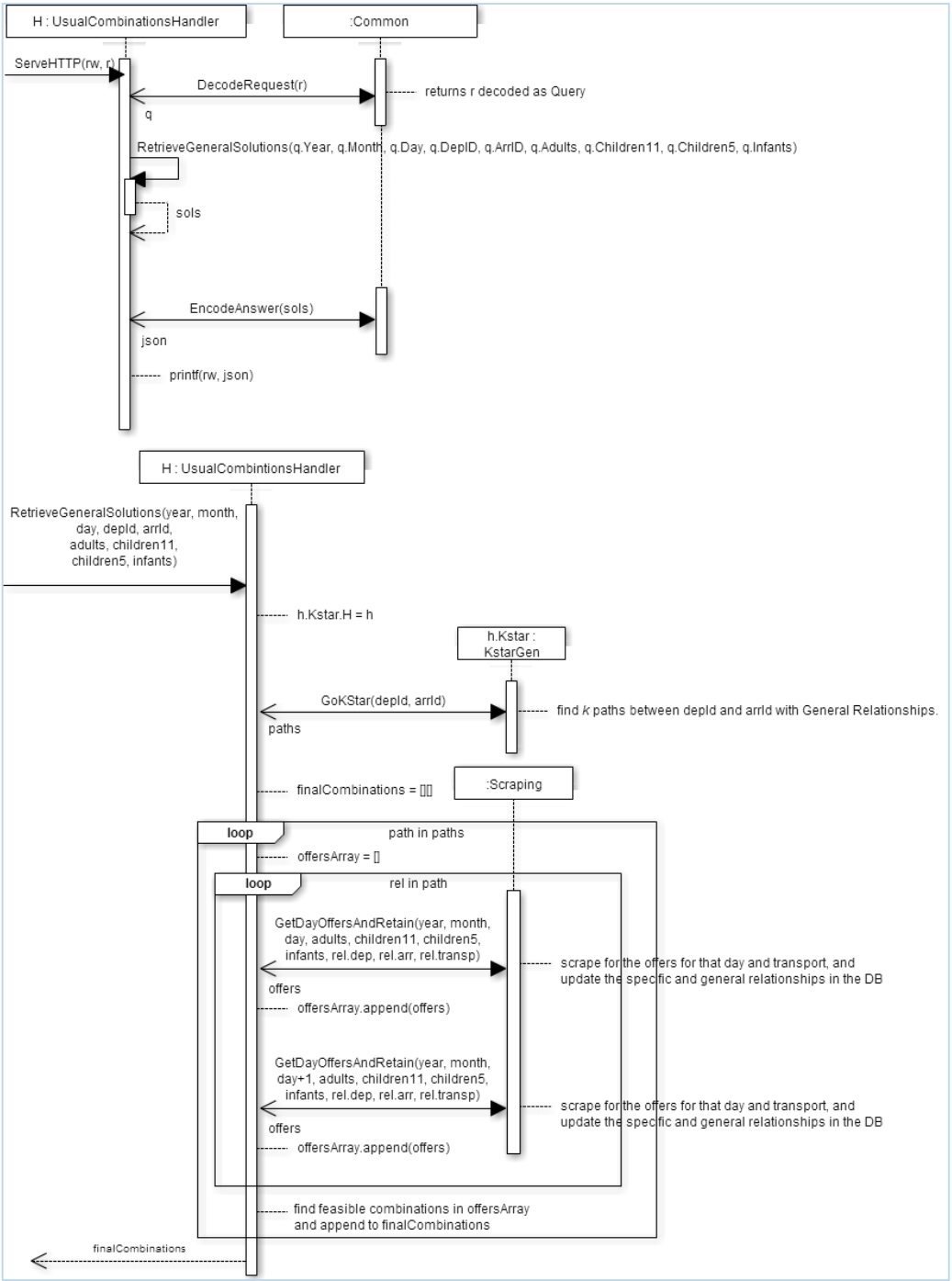

Figura 12. Diagrama de seqüència *UsualCombinations*.

5.3 DISSENY

5.3.1 DISSENY DE L'ARQUITECTURA: MVC

Com s'ha deixat entreveure al diagrama de flux del sistema (Figura 7), el nostre sistema utilitza el patró de disseny MVC (Model-Vista-Controlador) per a la divisió d'aquest en capes. La Vista serà la capa que interaccionarà amb l'usuari i estarà representada per les diferents pàgines web que se li mostrin, responsables de mostrar l'estat del Model a l'usuari i d'interaccionar amb aquest per modificar-ho; estarà implementada íntegrament al *front-end* de l'aplicació. D'altra banda, el Model serà el responsable d'emmagatzemar les dades i treballar amb elles; aquesta capa haurà de respondre correctament a les peticions de l'usuari, ja sigui enviant a la Vista pàgines webs estàtiques, o realitzant els càlculs necessaris i responent-li amb els resultats. Finalment, el Controlador serà l'encarregat de comunicar el Model i la Vista, traspassant les peticions de la Vista al Model, o les respostes del Model a la Vista. Tant el Model com el Controlador estaran implementats al *back-end* de l'aplicació.

5.3.2 DISSENY DE LA INTERFÍCIE

En aquest apartat es presentarà el disseny de la interfície d'usuari que s'ha ideat per a la pàgina web. Es mostrarà, per a les dues pàgines dissenyades – la d'introducció de dades al formulari i la de visualització dels viatges -, quines accions pot prendre l'usuari i com respon a aquestes accions la interfície, seguint l'ordre lògic d'accions que faria un usuari qualsevol per trobar els viatges desitjats. S'inclouen referències numèriques a les imatges per poder referir-nos a les accions des del text.

Primer de tot es mostra la pantalla d'introducció de les dades del viatge desitjat a un formulari, a la Figura 13. A continuació es comenten quines accions pot prendre l'usuari dins d'aquesta pantalla.

- Quan l'usuari fa clic a algun dels camps textuals per introduir la ciutat de sortida o la d'arribada (1), apareix una llista a sota d'aquests amb tot el conjunt de ciutats del sistema. Llavors, l'usuari pot escriure dins del camp per filtrar així la ciutat desitjada del conjunt de ciutats.
- L'usuari pot incrementar el valor del nombre d'adults, però només fins a 9. Al fer clic al botó de *"Incluir niños"* (3), s'obrirà un *popover* on es podrà incrementar la quantitat de nens, que es troben dividits en tres categories segons la seva edat: de 5 a 11 anys, de 2 a 4 anys, i de 0 a 1 any (nadons). Com es pot veure a (4), no es poden afegir més nadons que adults, i, si el nombre de nens d'entre 0 i 5 anys es superior que el d'adults, es mostra un missatge al mateix *popover* indicant que les empreses d'autobús no permeten això i que d'aquesta manera no es tindran en compte els viatges amb autobús a la cerca.
- Fent clic al camp textual per introduir la data de sortida (5), s'obre un calendari on l'usuari pot escollir el dia de sortida.
- Finalment, fent clic a *"Buscar"* (6), s'envien les peticions al servidor, i s'accedeix a la segona interfície.

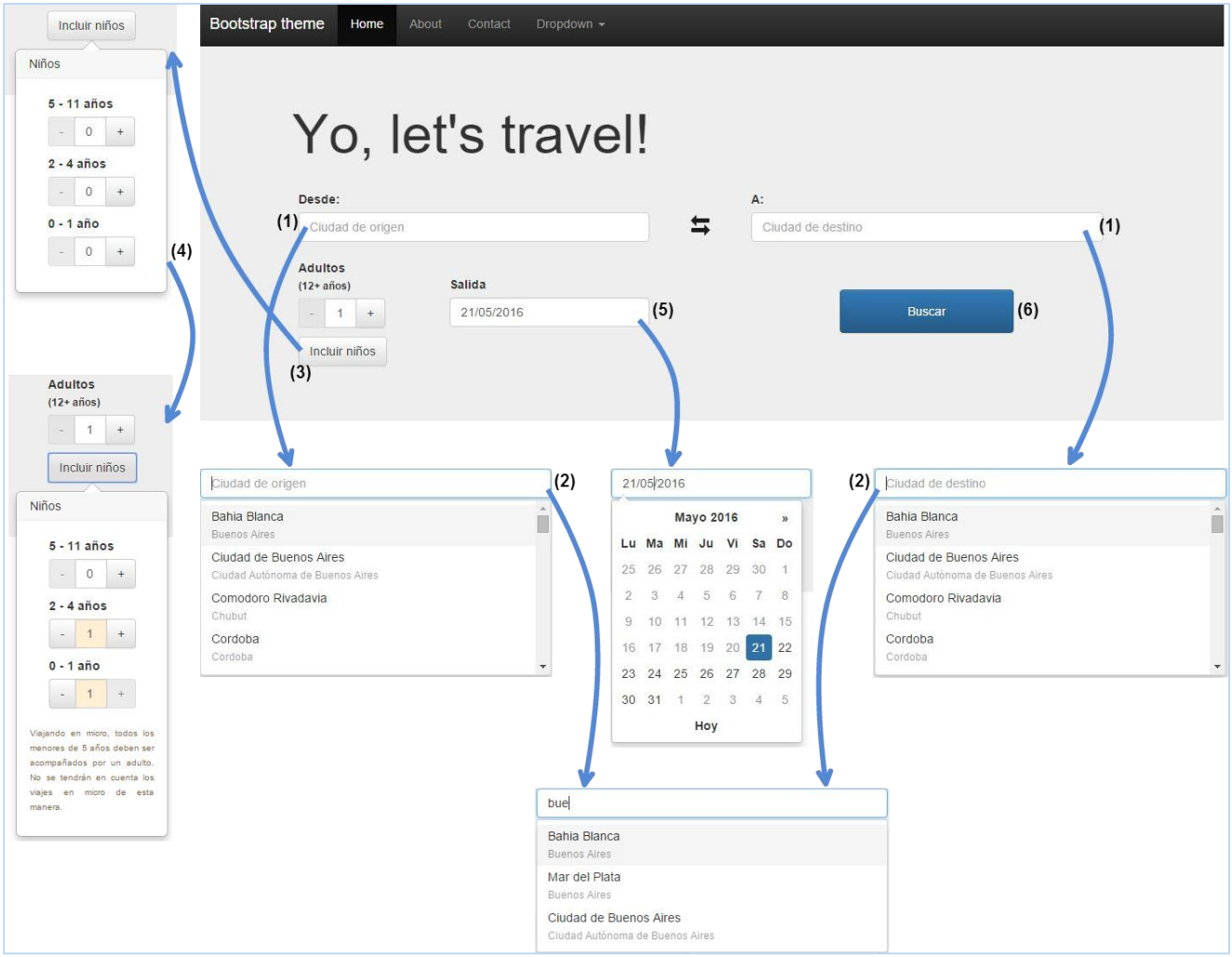

Figura 13. Disseny de la interfície - pantalla inicial.

Passem a comentar la segona interfície. A la Figura 14 es mostra l'estat d'aquesta just després que l'usuari és redirigit quan prem el botó de "Buscar". Mentre encara no hagin aparegut resultats, per a que l'usuari no pensi que el sistema s'ha "trencat", una animació li indica que encara s'estan buscant. A la Figura 15, finalment, es mostra l'estat de la segona interfície quan ja s'han mostrat alguns dels resultats, encara que no tots. Es comenten a continuació els elements de la pàgina i les accions que pot prendre l'usuari a aquesta:

- Primer de tot, al panell superior esquerra, es mostren les ciutats de sortida i d'arribada (1), juntament amb la data de sortida, que compta amb dos botons, a esquerra i a dreta respectivament, per incrementar o disminuir la data amb un dia (2); clicar un d'aquests botons buidaria la llista de viatges i tornaria a executar les peticions per obtenir-ne els resultats per a la nova cerca. A més, l'usuari pot clicar al botó *"Cambiar búsqueda"* per realitzar una altra cerca canviant tots els paràmetres (3); al clicar el botó, s'obrirà un nou panell on podrà canviar els paràmetres, utilitzant components amb un comportament similar als de la primera interfície. Si s'acaba realitzant la nova cerca, la web tornarà a l'estat de la Figura 14, canviant els textos necessaris, i sense necessitat d'actualitzar la pàgina.
- La barra superior (4), dona informació a l'usuari sobre l'estat de la cerca, amb el número de resultats trobats i una barra de càrrega que es va omplint a mesura que van arribant les respostes a les peticions (7), mostrant els missatges de *"Buscando enlaces directos...(1/3)"*, *"Buscando enlaces directos...(2/3)"*, *"Buscando enlaces directos...(3/3)"* i *"Buscando combinaciones...."*. A més, a la dreta es mostren les opcions d'ordenació de resultats (5), amb les opcions de *"Precio total"*, *"Precio por adulto"*, *"Salida"*, *"Llegada"* i *"Escalas"*; al canviar de criteri d'ordenació, l'ordre dels viatges mostrats canvia sense cap temps d'espera.
- Al panell esquerra (6), es troben les opcions per filtrar els viatges: per preu màxim, hora de sortida, dia d'arribada i número màxim d'escales. Al igual que amb l'ordenació, al canviar els filtres, els resultats s'expandeixen o es redueixen sense cap tipus d'espera.
- En quant als resultats, cadascun es mostra amb un panell (8) on s'indiquen l'hora i el dia de sortida i d'arribada, el nombre, el tipus i l'ordre dels transports, a més del preu total del viatge i del preu per adult. Clicant al botó *"Ver"* (9), l'usuari tindrà disponible el panell amb la informació sobre els transports, representat a la Figura 16.

Com s'ha dit, a la Figura 16 es mostra el panell amb la informació sobre els transports d'un resultat. Cada transport del resultat es representa amb una fila, en l'ordre corresponent, que inclou la següent informació: el tipus de transport i l'identificador del trajecte (el número de vol i l'aerolínia o el nom de l'empresa d'autobús i la categoria del seient), la ciutat i l'aeroport (o parada d'autobús) de sortida i d'arribada, l'hora de sortida i d'arribada, el preu total i el preu per adult, i un botó "Comprar", que quan es clica obre una nova finestra a la pàgina web d'on es poden adquirir els bitllets.

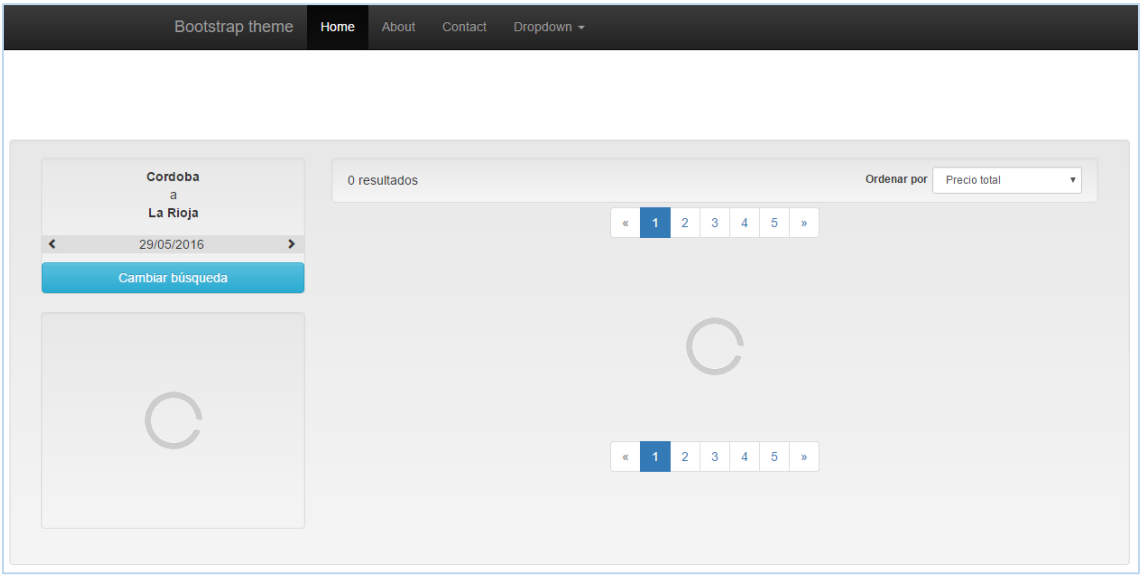

Figura 14. Disseny de la interfície - pantalla de resultats (carregant).

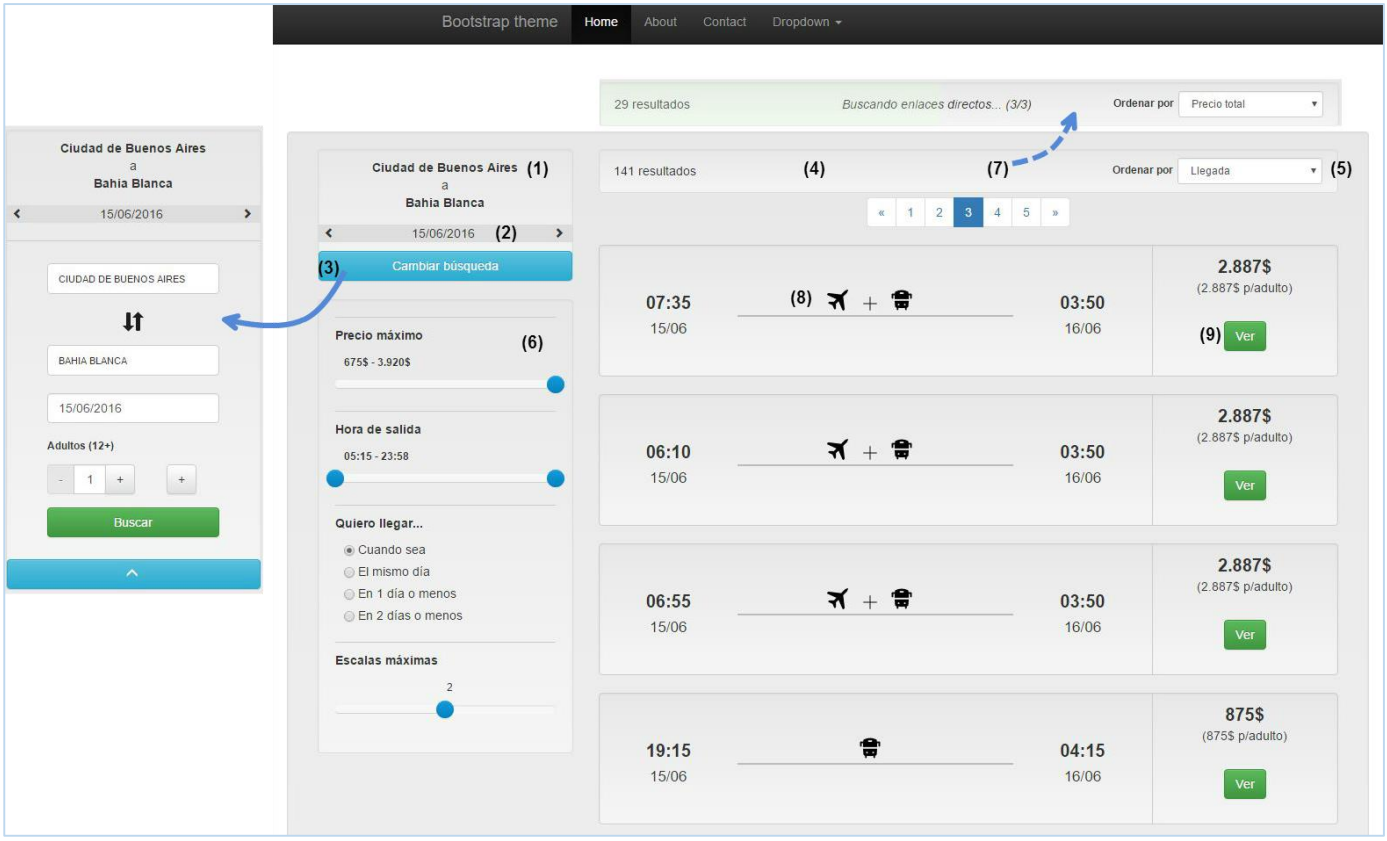

Figura 15. Disseny de la interfície - pantalla de resultats.

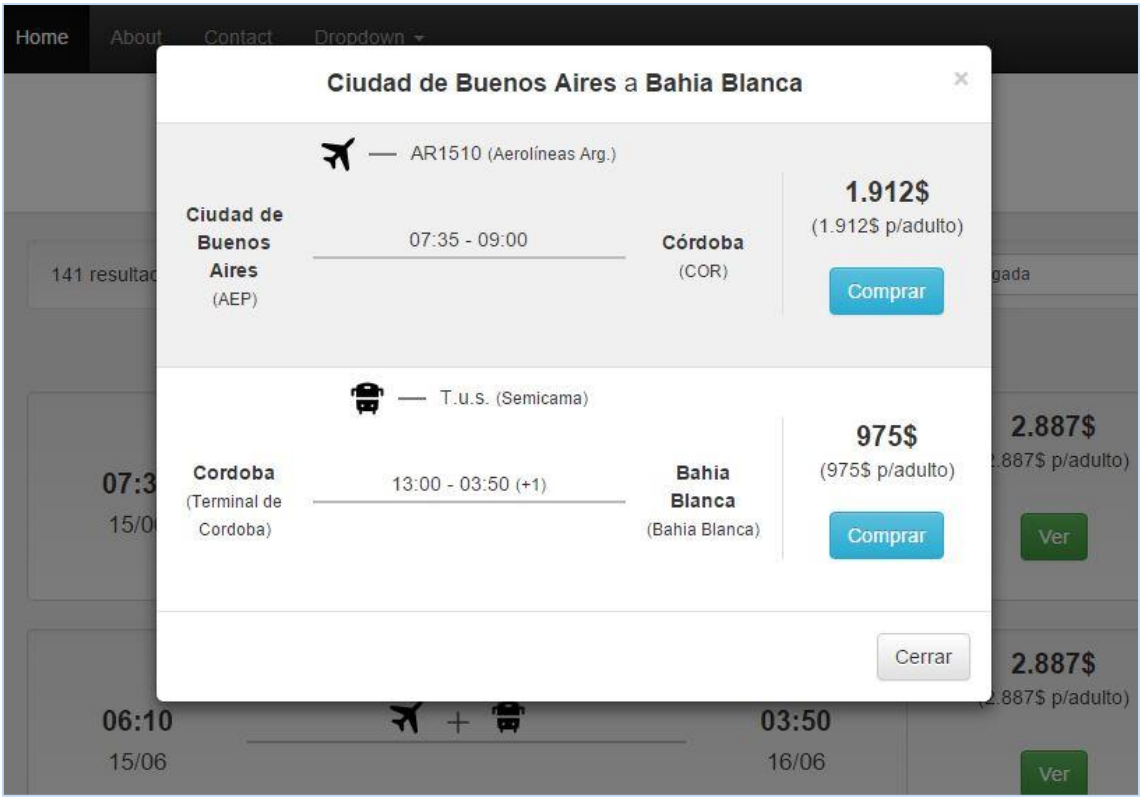

Figura 16. Disseny de la interfície - informació sobre transports.

6. IMPLEMENTACIÓ DE L'APLICACIÓ

Un cop explicat el disseny de l'aplicació, dins d'aquest apartat es parlarà sobre com s'ha decidit implementar aquesta. Primer de tot introduirem les eines i la tecnologia utilitzades per a la implementació, i després començarem a explicar, amb detall, tot el procés d'implementació.

6.1 TECNOLOGIA I EINES UTILITZADES

Seguidament es comentarà quines eines i tecnologies s'han escollit per desenvolupar el *back-end* del sistema, tenint en compte les característiques que tenia la aplicació i les funcionalitats que havia de complir. Concretament, es comenta l'elecció de la base de dades i del llenguatge de programació. Per a cadascun, s'explicarà per quins motius s'ha escollit i es detallaran les seves característiques principals.

6.1.1 LA BASE DE DADES

Com ja se sap, el sistema utilitza una base de dades orientada a grafs, amb l'objectiu de guardar-hi nodes com a ciutats i relacions com a trajectes entre aquestes. L'oferta de bases de dades orientades a grafs és força àmplia, i, tot i que no hi ha cap que destaqui clarament respecte a les altres ni en funcionalitat ni en nombre d'usuaris, es va decidir escollir Neo4j, ja que semblava ser una de les més actives – amb la última actualització fa uns mesos – i tenia força informació a internet, tant de la seva pròpia documentació com de fòrums externs. A més, compte amb una versió stand-alone molt simple d'utilitzar i amb una interfície basada en web força amigable, que permet realitzar-hi qualsevol transacció i visualitzar i explorar el graf de manera gràfica, i amb un llenguatge de transaccions declaratiu propi, Cypher, que segueix l'estil de comandes de SQL, però aplicades als grafs.

Apart d'aquestes particularitats, la seva característica més especial és que és accessible des de qualsevol llenguatge de programació, mitjançant la seva *API* web *REST*[10] integrada. Pel nostre cas particular, pel llenguatge de programació *Golang* (veure següent apartat), hi ha un projecte *open-source*[11] que integra aquesta *API* dins d'un paquet que permet realitzar les crides a la base de dades d'una manera molt més senzilla i intel·ligible.

6.1.2 EL LLENGUATGE DE PROGRAMACIÓ

A qualsevol projecte informàtic, escollir el llenguatge de programació més adequat a les característiques d'aquest és indispensable per assegurar l'eficiència del codi – i del sistema, en general – i aconseguir una implementació el més simple possible.

Per al nostre projecte, les característiques principals que ens interessaven del llenguatge de programació eren tres:

- Que fos un llenguatge *server-side*, és a dir, orientat a la implementació del codi executat a la banda del servidor d'una aplicació web.
- Que fos compilat (o si més no, amb un intèrpret eficient). Es buscava la màxima eficiència per part del llenguatge, i per tant no interessava un llenguatge interpretat com *Python* o *PHP*, ja que això afegiria una càrrega temporal innecessària, i no aprofitaríem cap dels avantatges de la interpretació.
- Que permetés la paral·lelització de manera eficaç i simple. Tenint en compte que el nostre programa hauria d'executar força peticions a servidors externs a cadascuna de les execucions, mitjançant els *scrapers*, l'execució d'aquestes de manera seqüencial seria terrible pels temps d'execució, i s'havia d'apostar per un llenguatge que permetés una paral·lelització de comandes de manera eficaç, o de forma nativa, o amb paquets externs.

[10] *Representational State Transfer*
[11] *Neoism* - github.com/jmcvetta/neoism

No va caldre una cerca gaire exhaustiva per trobar el llenguatge més adient per satisfer les nostres necessitats. Aquest era, sens dubte, *Golang* de *Google* [12], un llenguatge de programació *open-source*, desenvolupat per la companyia i per diversos contribuïdors des de l'any 2007. El llenguatge ha adquirit una gran fama als últims anys i ja s'ha fet un lloc entre els llenguatges més utilitzats i valorats per a dissenyar aplicacions a la banda del servidor.

És un llenguatge compilat, de tipat estàtic (seguint la tradició de C), i que utilitza un *garbage collector* per mantenir neta la memòria. Independentment, la característica principal d'aquest llenguatge és la seva paral·lelització nativa de rutines: afegint la comanda *go* davant de qualsevol mètode, es fa que aquest sigui executat per un altre *thread* diferent del sistema, de manera paral·lela. A més, i com és d'esperar, incorpora una llibreria de sincronització que incorpora elements bàsics de sincronització com són els *Mutex* (per esperar a tots els processos creats a un mateix punt), els *Locks* (per evitar l'accés simultani a una variable) o els *Channels* (per comunicar els *threads* entre sí).

A més d'aquestes característiques, el llenguatge compta amb altres d'interessants, com per exemple:

- Compta, en general, amb excel·lents paquets, sobretot per al desenvolupament de les tasques pròpies d'un servidor; per exemple, per a la recepció i l'enviament de paquets *HTTP*, fent molt senzill implementar un servidor web operatiu que escolti els ports corresponents a l'espera de peticions externes.
- Té una sintaxi molt estètica i senzilla de llegir, a vegades propera a la dels llenguatges dinàmics o d'*scrpiting*. Com a exemples, utilitza la inferència de tipus a l'hora d'assignar valors a variables, l'opció de retorn de múltiples valors per part de les funcions, la utilització d'una única declaració pels bucles (*for*), la declaració d'una nova variable amb ':=' i la reescriptura amb '=', o la no necessitat d'afegir un punt i coma al final d'una línia de codi.
- No té una entitat definida explícitament com a "classe", ja que *Golang* no té orientació a objectes explícita. L'orientació a objectes, existeix, però no és tan clara com a la majoria de llenguatges. Les classes es defineixen a partir d'*structs*, i l'herència de classes es pot fer, o bé a partir de la *incrustació* d'un *struct* dins d'un altre, o bé a partir d'interfícies.

LLIBRERIES EXTERNES QUE S'UTILITZEN

Per matisar l'explicació del llenguatge, a continuació es citen les llibreries (o paquets) que s'utilitzen d'aquest per a fer-nos la vida més fàcil a l'hora d'implementar el codi. De les pròpies de *Golang*, se n'utilitzen les següents:

- *net/http*, que proporciona implementacions per a clients i servidors *HTTP* (l'exemple comentat abans).
- *sort*, que implementa les funcions d'ordenació de llistes.
- *strconv*, que permet realitzar conversions des de o cap a *strings.*
- *sync*, que proporciona les eines esmentades de sincronització de *threads*.
- *time*, que implementa la funció *Date*, que ajuda a realitzar càlculs amb dates.
- *Conteiner/heap*, que implementa els *binary heaps*, molt útils, no només per a crear aquest tipus d'estructura sinó per implementar llistes de prioritat amb ells.
- *os*, que proporciona una interfície per a realitzar funcionalitats de sistema operatiu.
- *encoding/json*, que ajuda a codificar i descodificar objectes *JSON*[12].

De llibreries externes, només se n'utilitza la comentada anteriorment per interactuar amb la base de dades d'una manera més còmode: github.com/jmcvetta/neoism.

Evidentment, *Golang* només és el llenguatge de programació utilitzat per a la construcció del *back-end*, que és el que es comentarà a aquest apartat d'Implementació. Pel *front-end*, s'ha utilitzat, com és habitual, *Javascript* amb el seu *framework jQuery* i les llibreries de classes d'estil *Bootstrap* per dissenyar amb més facilitat el document *html*.

[12] *Javascript Object Notation.*

6.2 DETALLS DE LA IMPLEMENTACIÓ

Dins d'aquest apartat explicarem com s'han implementat els diferents mòduls de l'aplicació, i quina complexitat han acabat tenint.

Dels cinc mòduls generals de l'aplicació, definits als diagrames de seqüència de l'apartat 5.2 (el càlcul dels viatges directes amb Aerolíneas, LAN i Plataforma10, i el càlcul de les combinacions amb relacions Específiques i amb relacions Generals), podem extreure dos clars sub-mòduls principals: el de la cerca amb K* dels *k* camins òptims, que té la versió per a les relacions Específiques i la versió per a les relacions Generals; i el de la transformació dels camins trobats per K* a un conjunt de viatges reals amb combinacions llestos per ser retornats al controlador. Aquest segon, inclou a la vegada els sub-sub-mòduls d'*scraping* (cridant a la funció *GetDayOffersAndRetain*) i de cerca de les combinacions factibles a partir de les ofertes obtingudes; el primer, és també l'únic que s'utilitza als mòduls de càlcul de viatges directes.

Llavors, dividirem l'explicació de la implementació en aquests dos sub-mòduls, dintre dels quals explicarem els sub-sub-mòduls corresponents. Primer de tot explicarem el sub-mòdul de K* i seguidament el de la transformació dels camins.

6.2.1 K*

Per explicar aquest mòdul, primer s'explicarà el funcionament de K* i les seves característiques temporals i espacials, i, seguidament, s'explicarà la implementació al nostre projecte, tant a la versió per a les relacions Generals com per a les relacions Específiques.

L'ALGORISME K*

K* [9], és un algorisme de cerca dirigida per cercar els k camins més curts entre una parella de vèrtexs *s* i *t* donada dins un graf dirigit G amb pes a les arestes. L'algorisme K* té dues característiques fonamentals: la primera, és que opera *on-the-fly*, és a dir, no requereix que el graf estigui disponible explícitament a memòria principal, sinó que s'utilitzaran (i s'obtindran de la base de dades en el nostre cas) les porcions de graf que siguin necessàries a mesura que s'avanci en la cerca; i la segona, que es guia a partir de funcions heurístiques. Tot això fa que l'algorisme tingui una complexitat teòrica – utilitzant una heurística consistent -, tant temporal com espacial, de *O(m + nlogn + k)*, on *n* són el nombre de vèrtexs del graf i *m* el nombre d'arestes, que iguala la complexitat de l'algorisme d'Eppstein [10], considerat històricament el millor per la tasca. Tot i tenir la mateixa complexitat teòrica, l'algorisme K* és força superior a la pràctica, sempre i quan s'utilitzin les heurístiques adequades; i a més a més, estalvia una gran quantitat de memòria al no requerir tot el graf en memòria. La idea principal de l'algorisme és alternar l'execució de l'algorisme de cerca A* i el de Dijkstra. El primer s'executarà amb l'objectiu de descobrir nous camins dins del graf G que arribin fins al vèrtex de destí *t*, i el segon s'aplicarà sobre un segon graf generat amb informació relativa a les noves arestes trobades per A* (anomenat *P(G)*, o *path graph*), i serà l'encarregat de descobrir els camins òptims des de *s* fins a *t*; A* es tornarà a executar (sempre i quan no s'hagi explorat ja tot el graf) quan Dijkstra es quedi sense nous camins per explorar, amb l'objectiu de descobrir noves parts del graf. Aquesta alternança entre els algorismes acabarà, juntament amb l'algorisme, quan Dijkstra hagi trobat els *k* camins òptims, o quan no es puguin trobar més camins.

Abans de començar l'explicació en detall de l'algorisme, cal definir els conceptes de *tree edge* i de *sidetrack edge*. Dins un graf dirigit amb pesos *G*, donat un vèrtex *u* pertanyent a *G*, existeix un arbre de cerca *T* per aquest vèrtex, format per tots els vèrtexs accessibles des de *u*, que compleix que per tot vèrtex *v* diferent a *u* dins de l'arbre, la distància entre *u* i *v* a l'arbre és igual a la mínima distància (entenent distància com la suma de pesos de les arestes del camí) entre *u* i *v* a *G*. Llavors, podem assegurar que qualsevol aresta *(u,v)* – amb *u*, *v* vèrtexs qualssevol de G - , o forma part d'aquest arbre de cerca *T*, o no en forma part. Si en forma part, diem que és un *tree edge*, i si no en forma part, diem que és un sid*etrack edge*. A la Figura 17 s'intenta il·lustrar el concepte. En línies contínues es mostren els *tree edges* de l'arbre de cerca de *s0*, i en línies discontínues els *sidetrack edges*.

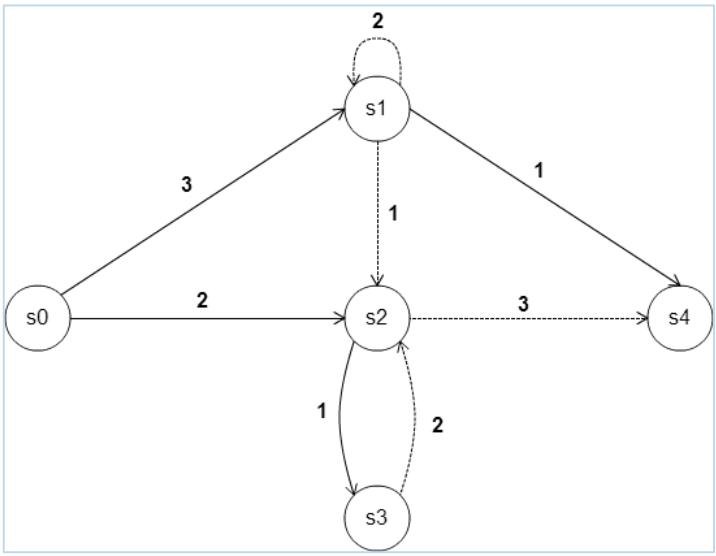

Figura 17. Representació de *sidetrack edges* (línies discontínues) i *tree edges* (línies contínues) a un graf - Extret de [9].

A més a més, un cop entesos aquests dos conceptes, podem introduir el concepte de cost de la desviació (o *detour cost*). Per a qualsevol aresta del graf *(u,v),* la funció de desviació *δ(u, v)* representa el cost extra que suposa arribar a v des de u, en comptes d'arribar-hi mitjançant el camí mínim de s fins a v definit a l'arbre de cerca de s. Llavors, $\delta(u, v) = g(u) + c(u,v) - g(v)$, on *g(x)* representa el cost del camí *s-x* a l'arbre de cerca (és a dir, el cost del camí mínim entre *s* i *x*), i *c(x,y)* representa el pes de l'aresta *(x,y)*. Observant novament la Figura 17, veiem com si intentem arribar a *s4* des de *s2*, tindrem un cost de desviació δ(s2, s4) = 2 + 3 – 4 = 1; i si intentem arribar a *s2* des de *s3*, tindrem un cost de desviació $\delta(s3, s2) = 3 + 2 - 2 = 3$.

Com hem dit, l'algorisme construeix un segon graf *P(G)* format a partir del graf *G* (o més ben dit, de la part d'aquest trobada fins al moment per A*). Aquest graf *P(G)*, en termes molt generals, està format per múltiples *binary heaps*[13] interconnectats entre sí. Més específicament, per cada vèrtex del graf explorat fins al moment, es generen dos tipus de *binary heap*, els *Hin* i els *Ht*. Llavors, per cada vèrtex *v* disponible, tindrem:

- Un $H_{in}(v)$ (*incoming heap*), amb un vèrtex per cada *sidetrack edge* incident a *v*, amb un valor igual al cost de desviació d'aquesta aresta. El *heap* estarà ordenat per aquests

[13] Un *binary heap* és un arbre binari (2-ari) on, per cada node, es compleix que els seus fills tenen un pes major o igual al seu, i a més, el fill esquerra té un pes major o igual al fill dret.

valors. L'arrel de l'arbre tindrà com a molt un fill, i l'anomenarem $root_{in}(v)$. A la Figura 18 es mostren els *incoming heaps* derivats del graf de la Figura 17.

- Un $H_T(v)$ (*tree heap*), que es construeix de la manera següent. Si v és el vèrtex d'origen (és a dir, $v = s$), $H_T(v)$ serà simplement un *binary heap* buit, al qual se li afegirà $root_{in}(v)$ – si $H_{in}(v)$ no és buit, clar -. Si v no és el vèrtex d'origen, anomenem u al pare de v dins de l'arbre de cerca T. Llavors, podem pensar en $H_T(v)$ com a una còpia de $H_T(u)$, a la qual se li afegeix $root_{in}(v)$. Si $H_{in}(v)$ és buit, llavors, $H_T(v)$ és idèntic a $H_T(u)$. L'arrel d'un *tree heap* pot tenir un o dos fills, i l'anomenarem $R(v)$. A la Figura 19 es mostren els *tree heaps* derivats del graf de la Figura 17.

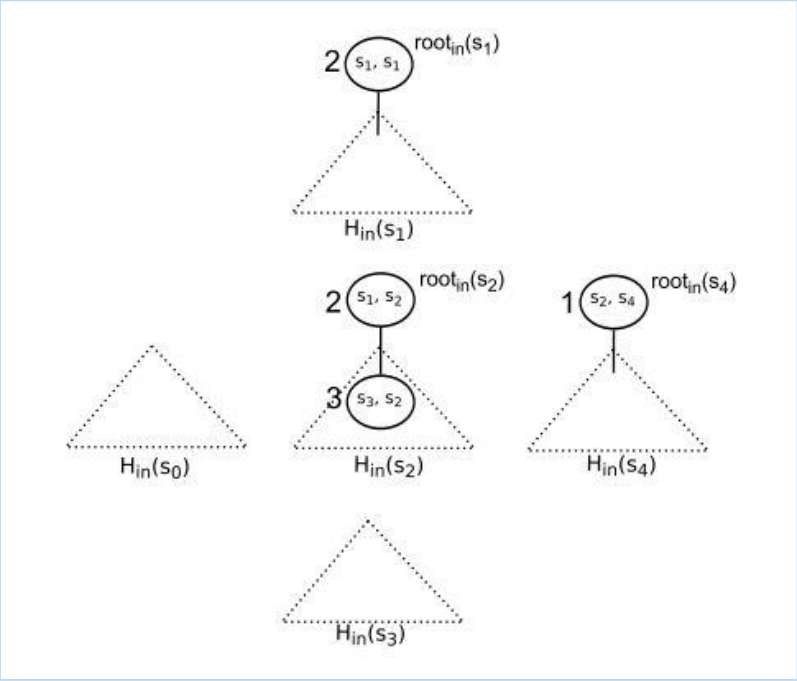

Figura 18. *Incoming Heaps (Hin)* derivats de la Figura 17 - Extret de [9].

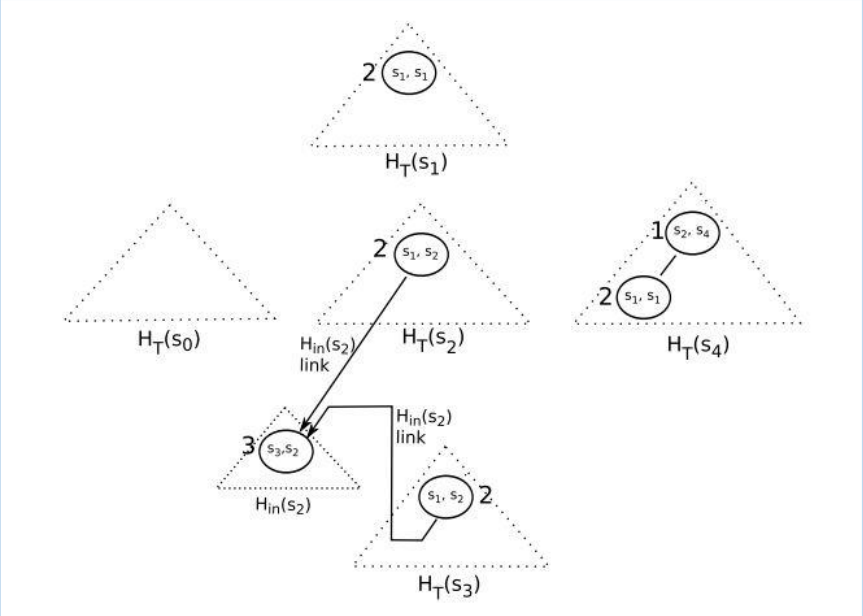

Figura 19. *Tree Heaps (Ht)* derivats de la Figura 17 - Extret de [9].

Donades aquestes dues definicions, ens referim a *heap edge*, quan parlem d'una aresta originada dins d'un *tree heap* o d'un *incoming heap*.

L'estructura final de *P(G)* vindrà donada de la següent manera. Per cada node *n* de *P(G)* – pertanyent a un *incoming heap* o a un *tree heap* – que representi una aresta *(u,v)*, li afegim un punter cap a *R(u)*, és a dir, l'arrel de *Ht(u)*. Aquestes arestes s'anomenaran *cross edges*. També s'afegeix un node especial *R* amb una única aresta *cross edge* apuntant cap a *R(t)*.

A més a més, es defineix una funció Δ sobre les arestes de *P(G)*, que definirà el pes d'aquestes. Donat una aresta *(n,n')* a *P(G)*, associada a les arestes de *G*, *e* i *e'*, respectivament, definim $\Delta(n,n')$ com:

- $\Delta(n,n') = \delta(e') - \delta(e)$, si *(n,n')* es tracta d'un *heap edge*, o
- $\Delta(n,n') = \delta(e')$, si *(n,n')* es tracta d'un *cross edge*.

A la Figura 20 es mostra la forma final de *P(G)* derivada del graf de la Figura 17.

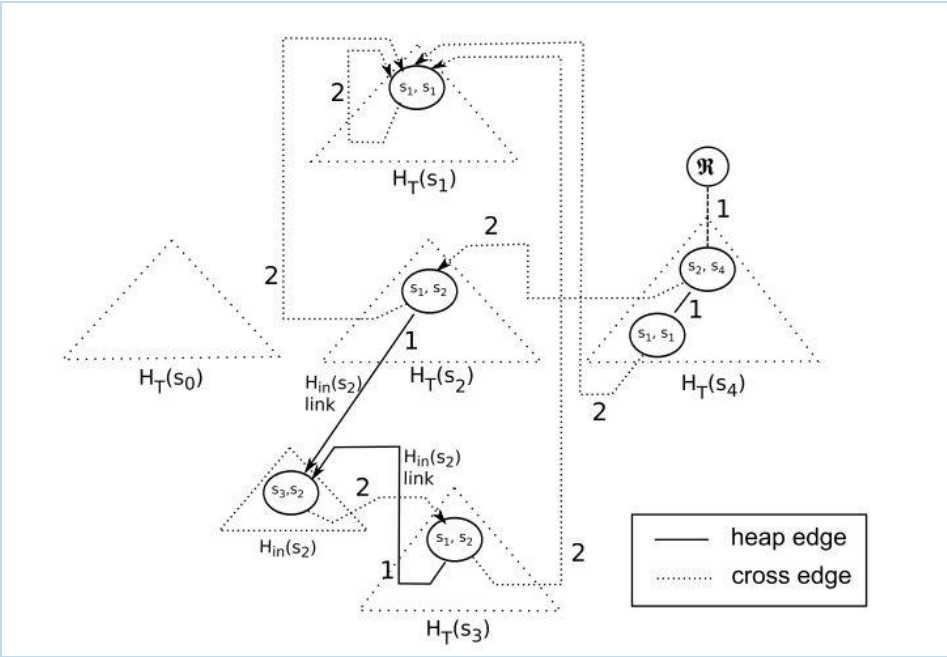

Figura 20. ***Path Graph (P(G))*** **derivat del graf de la figura 17 - Extret de [9].**

Com sabem, Dijkstra s'executarà sobre aquest *path graph* i trobarà camins dins d'ell. Cada camí trobat a aquest graf suposarà un nou camí al graf original (Corol·lari 1 de [9]); i, a més, cada nou camí trobat per l'algorisme serà el camí més curt possible dels que queden per explorar, sempre i quan l'heurística d'A* sigui admissible (Teorema 2 de [9]). A continuació s'explica com passar d'un camí trobat per Dijkstra a *P(G)* a un camí real del nostre graf original. Sigui $\sigma = n_0 \rightarrow \cdots \rightarrow n_r$ un camí dins de *P(G)*, cada *cross edge* (n_i,n_{i+1}) representa la selecció del *sidetrack edge* associat a n_i (i també si n_i és l'últim node), mentre que cada *heap edge* (n_i,n_{i+1}) representa la selecció del *sidetrack edge* associat a n_{i+1}. Llavors, aconseguirem la seqüència *seq(σ)* de *sidetrack edges* de la següent manera: primer s'afegeix l'aresta associada a l'últim node de σ, és a dir, n_r; després, s'iteren els nodes de σ des de l'últim fins al primer, i, per cada *cross edge* (n_i,n_{i+1}) que es trobi, amb n_i diferent a *R*, s'afegeix l'aresta associada a n_i.

Un cop obtingut *seq(σ)*, només cal recórrer el graf *G* des del vèrtex de destí *t* fins al node d'origen *s*, afegint al final del camí final la següent aresta a l'arbre de cerca *T*, sempre i quan el node destí de l'última aresta de *seq(σ)* no coincideixi amb l'últim node afegit al camí final. Si coincideixen, s'afegeix l'aresta en qüestió, i s'elimina de *seq(σ)*.

Un cop explicada tota l'estructura de dades darrere de K*, passem a explicar el flux d'execució de l'algorisme, que es mostra a la Figura 21. Es pot veure un exemple de pseudocodi a l'apartat 4.4. de [9].

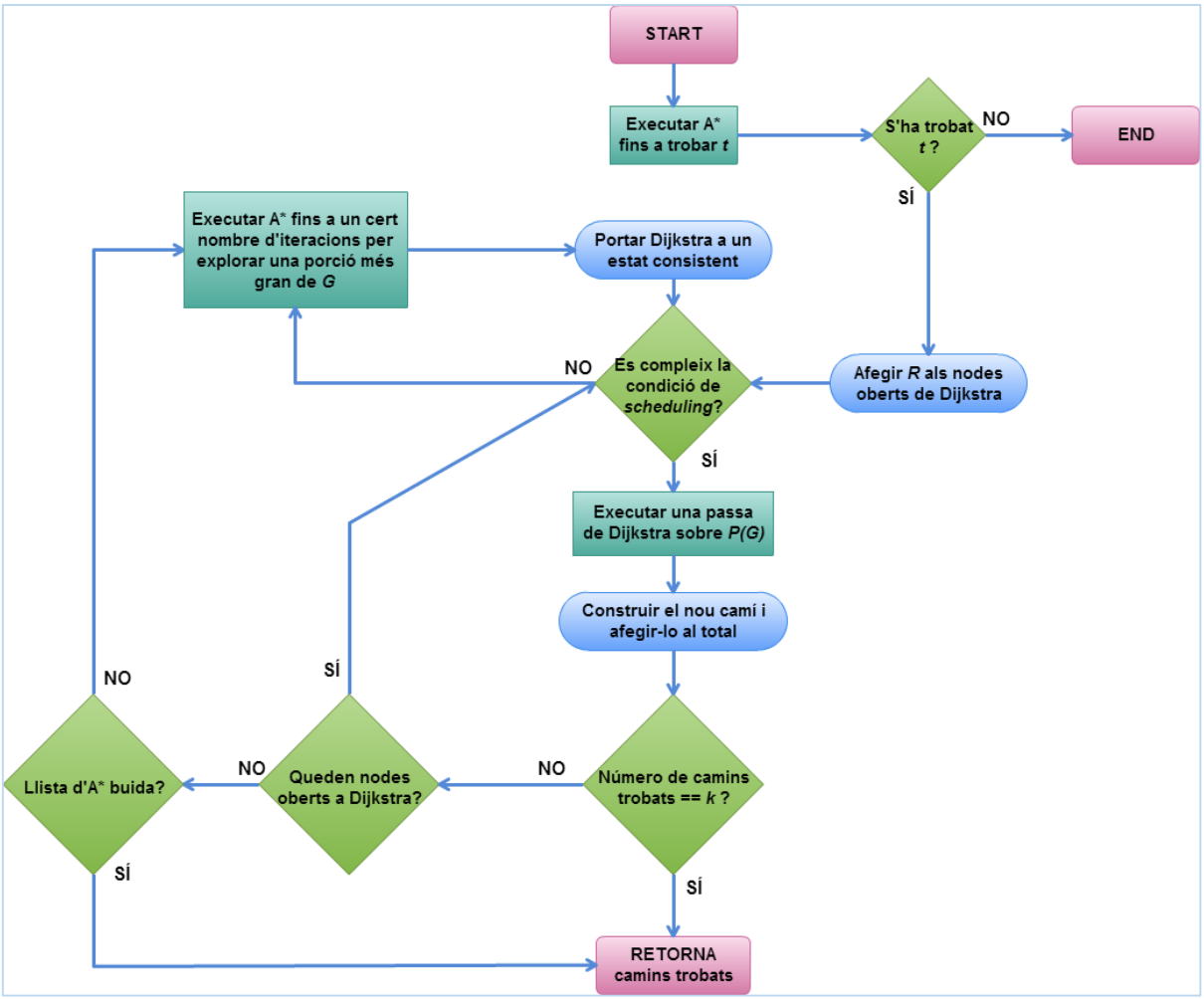

Figura 20. Flux d'execució de K*.

L'algorisme comença executant A* sobre el graf *G* fins a trobar el node de destí *t* des del node d'origen *s*. Si no es troba, vol dir que no hi ha camins possibles, així que s'acaba l'algorisme. Si es troba, s'afegeix el node *R* de *P(G)* a la cua de prioritat de Dijkstra. Llavors, per continuar amb Dijkstra, s'ha de veure si es compleix una certa condició de *Scheduling* (sempre i quan hi hagin nodes a la cua d'A*), que es basa en comprovar el següent:

- Sigui *u* el primer node de la cua de prioritat d'A* i *n* el primer node de la cua de prioritat de Dijkstra.

- $d := max \{ d(n) + \Delta(n,n') \mid n' \in successors\ de\ n \}$

- $g(t) + d <= f(u)$?

Essent *d(n)* distància enregistrada a *Dijkstra* per *n*, *g(t)* el camí més curt des de *s* fins a *t*, i *f(u)* és la funció d'estimació del cost d'A* (*g(u) + h(u)* , cost conegut des de *s* fins a *u* més cost heurístic des de *u* fins a *t*).

Si la condició és certa, es procedeix amb l'algorisme de Dijkstra. Si no ho és, es torna a executar A*. Un cop executada la nova passa de Dijkstra (que fa les operacions habituals, recórrer els fills de *n* afegint-los a la cua de prioritat amb la distància *d(n) + Δ(n,n')*, i treure *n* de la cua), es pot generar el següent camí òptim amb el mètode explicat abans. Si s'ha arribat als *k* camins òptims, es retornen; sinó, s'ha de mirar si es pot fer una nova passa de Dijkstra (és a dir, si queden nodes a la cua de nodes oberts). Si no es pot fer, es mira si la cua de nodes d'A* és buida o no; si és buida, vol dir que ja s'ha explorat tot el graf, i per tant, no es podran trobar més camins, i si no ho és, es torna a executar A* per ampliar el coneixement sobre *G*. El nombre d'iteracions que s'han d'executar d'A* quan es torna a cridar no és un valor exacte, però els autors recomanen explorar-ne el doble de nodes que s'han trobat fins al moment.

Una qüestió important que cal mencionar és la interdependència entre A* i Dijkstra. Quan Dijkstra ja no pot explorar més i es torna a cridar a A* per a que recorri més nodes al graf, s'ha de tornar a re-construir *P(G)*, ja que la nova cerca d'A* pot haver afegit, modificat o inclús eliminat nodes a *P(G)*. Això podria arribar a implicar que l'estat actual de la cerca de Dijkstra fos erroni, fins al punt que potser s'hauria de tornar a executar des de zero.

Tot i així, als Lemes 6 i 7 i als Corol·laris 2 i 3 de [9], s'assegura que si l'heurística utilitzada a A* és admissible, l'estat de la cerca a Dijkstra no es veurà modificat pas. I encara més, si l'heurística és consistent, no s'haurà ni de tan sols reconstruir *P(G)*. Llavors, és de vital importància que ens assegurem que l'heurística que s'utilitzi al nostre sistema (que s'explicarà més endavant) sigui, al menys, admissible.

LA IMPLEMENTACIÓ DE K*

Al present apartat ens encarregarem d'explicar la implementació de K* a la nostra aplicació. S'explicarà, en relació a les classes introduïdes a l'apartat de Disseny, amb quines estructures de dades es modela la implementació, i quin és el funcionament i la complexitat de les funcions involucrades. D'ara en endavant, quan es parli dels nodes *s* o *t*, s'estarà fent referència als nodes del graf de cerca; és a dir, als nodes ciutat d'origen i ciutat de destí en el cas de les relacions Generals, i als nodes *a* i *b* en el cas de les relacions Específiques.

S'ha de recordar, però, que existeixen dues implementacions de K*, una per a la cerca de camins amb relacions Específiques i l'altra amb relacions Generals. Aquestes dues implementacions no divergeixen en la lògica principal de l'algorisme, i s'han implementat com a dos mòduls separats per la diferència en els seus grafs d'entrada. Recordem que amb les relacions Generals s'utilitza el graf original, mentre que amb les relacions Específiques s'ha de "crear" un graf alternatiu amb els nodes representant les relacions del graf original (realment no es crea explícitament, sinó que simplement s'interpreta d'una manera diferent el graf de la base de dades). Aquesta diferència fa que la manera de treballar de l'algorisme amb els nodes i amb el graf en general sigui totalment diferent, i per tant l'opció més simple sigui realitzar aquestes dues implementacions per separat. Les diferències entre aquestes dues implementacions s'aniran explicant i entenent a mesura que s'expliqui l'estructura i la finalitat de cadascuna de les classes, ja que s'aniran matisant les característiques de cadascuna de les sub-classes corresponents a cada implementació en particular. D'ara en endavant quan es parli de *graf original* (o *G*), s'estarà fent referència al graf d'entrada de l'algorisme, és a dir, al graf present a la base de dades per als camins amb relacions Generals, i el graf alternatiu per als camins amb relacions Específiques.

Ambdues implementacions utilitzen les classes contenidores *Ht* i *Hin* per construir el *path graph* o *P(G)*, que utilitzen com a nodes la classe *Node*, i implementen la interfície *Heap* de *Golang* per mantenir una estructura de *binary heap* ordenat[14], implementant les funcions corresponents (*Len, Less, Swap, Push, Pop*), totes de temps constant. A més, implementen la funció *getNode(u,v)*, que, donats els identificadors d'un node, recorre el *heap* fins a trobar el node en qüestió; per tant, temps lineal en relació al nombre de nodes dins el contenidor.

Com s'ha dit, els nodes de *P(G)* s'identifiquen per dos valors, *u* i *v*, que corresponen als identificadors dels nodes de sortida i d'arribada de la relació (*sidetrack edge*) del graf original a la qual representen. La classe *Node*, a més, compta amb altra informació com el transport de la relació (*transp*), el cost de la desviació (*dValue*), l'identificador del *Hin* i/o del *Ht* al qual es troben (*hin, ht,* ja que un node pot estar només a dintre d'un *Hin*, o a dintre d'un *Hin* i d'un *Ht* a la vegada), la seva posició dins de l'*Ht* i de l'*Hin* (*htIndex*, *hinIndex*), la distància actual a l'algorisme de Dijkstra (*dist*), l'índex del pare dins del *heap* (*indParent*) i un booleà indicant si ha sigut obert per Dijkstra. Utilitza tres funcions de complexitat constant: *isTop()*, per saber si es troba a la cima de l'arbre, *isHin()*, per saber si és a únicament a dintre d'un *Hin*, i *getHtOrHin()*, per retornar l'índex del *heap* dins del contenidor de la classe *Graph*, que a continuació s'explica (del *Hin* si és només a un *Hin* o del *Ht* si és a dintre d'un *Ht*).

El conjunt d'*Hins* i d'*Hts* s'emmagatzema a dins de la classe *Graph* en dos *maps* diferents, indexats per l'identificador de cada node del graf original que en tingui. A més, implementa quatre funcions per obtenir els quatre possibles fills d'un node qualsevol dins dels *heaps*: *getRightHin*, *getLeftHin*, *getRightHt* i *getLeftHt*, totes de temps constant; i una altra funció, que utilitza les quatre anteriors, per obtenir tots els fills que tingui el node als *heaps* (és a dir, mitjançant *heap edges*): *succHeapEdges*.

Es creen dues sub-classes de la classe *Graph*: *GraphSpec* i *GraphGen*, per cadascuna de les implementacions, respectivament. Ambdues emmagatzemen tant l'estructura del graf *G* com la informació dels nodes de *P(G)* dins de l'algorisme de Dijkstra. L'estructura del graf s'emmagatzema a dins d'un *map* del tipus de node que correspongui al graf original. A *GraphGen*, com es treballa amb nodes equivalents a ciutats, s'emmagatzemen objectes de la classe *City*, que contenen l'identificador del node a la base de dades (*Id*, pel qual s'indexa el *map*), l'identificador del node predecessor dins de l'arbre de cerca – calculat per A* -

[14] A *Golang*, una classe implementa una interfície si implementa totes les seves funcions.

(*CameFrom*), i el tipus de transport amb el qual arriba aquest predecessor (*Transp*). A *GraphSpec*, els nodes representen relacions a la base de dades, i per tant, a més de l'identificador de la relació a la base de dades i l'identificador del predecessor (*Id* i *CameFrom*), tenim tota la informació referent a la relació: *DepTime*, *ArrTime*, *Weight*, *Transp*, *DepNode* i *ArrNode*. La informació dels nodes de *P(G)* dins de l'algorisme de Dijkstra s'emmagatzema en tres *maps* diferents: el dels nodes tancats per l'algorisme, és a dir, els quals els seus fills ja han estat afegits a la llista d'oberts (*Closed*), el dels nodes afegits a la llista d'oberts però encara no tancats (*Explored*), i el de la llista de pares d'un node (*Parents*)[15]. La indexació d'aquests *maps* és diferent segons la sub-classe. A *GraphSpec* s'indexen amb tres valors: l'índex de l'*Hin* o l'*Ht* al qual pertanyen, el vèrtex de sortida *u* i el vèrtex d'arribada *v*. A *GraphGen*, com que el seu graf original (el de la base de dades) és un *multigraph*[16] de fins a tres relacions per cada dos nodes en una direcció (una per cada tipus de transport), cal afegir als tres índexs anteriors el del tipus de transport de la relació.

Ambdues subclasses implementen els mateixos mètodes: *createHt(int)*, per crear el *Ht* del node demanat, obtenint el *Ht* del seu predecessor al arbre de cerca (si no existeix, cridant a *createHt* d'aquest), copiant els nodes dins d'aquest *Ht* i afegint-hi l'arrel del seu *Hin* (per tant, amb temps lineal respecte el nombre de relacions a *G*); i *SuccCrossEdges(Node)*, per obtenir els nodes accessibles mitjançant *cross edges* (amb temps constant).

Per mantenir l'execució global de l'algorisme s'utilitza la classe *Kstar*, que emmagatzema a *openD* els nodes oberts per Dijkstra i implementa la interfície *Heap* per mantenir aquesta variable com una cua de prioritat ordenada pel valor *dist* dels nodes. En quant a les sub-classes per a cada implementació – *KstarGen* i *KstarSpec* -, ambdues mantenen una variable *res* que va emmagatzemant els resultats que es troben per cada passa de Dijkstra a una llista (en el cas de *KstarGen* es retornen llistes de la tupla *[idNode][transport]* i en el cas de *KstarSpec* es retornen llistes d'identificadors de relacions), i, en concret, *KstarSpec* ha de mantenir informació sobre la data de sortida, per assegurar que no es tenen en compte viatges que comencin a una data diferent, i el rang de dates en el qual es permeten combinacions (variables *lookup* de la classe), per limitar la quantitat de relacions que es tenen en compte a A*.

Ambdues sub-classes implementen mètodes similars: *schedulingMechanismEnabled()*, per comprovar si el mecanisme de *scheduling* està activat (temps constant); *getTetaSeq(Node)* i

[15] Evidentment, com un node de *P(G)* pot ser indexat tant per un *heap edge* com per un *cross edge*, pot tenir més d'un pare.
[16] Un graf amb vàries arestes possibles entre dos nodes.

getPath, per aplicar la conversió entre camí de *P(G)* a camí de *G*, explicada amb anterioritat (temps lineal en relació al nombre de relacions a G); *dijkstraStep()*, per aplicar una passa de l'algorisme de Dijkstra; *resumeAstar(int)*, per reprendre la cerca a A* quan sigui necessari (cal passar-li la ciutat d'arribada, *t*); i *GOKStar(int,int)*, que actua com a "*main"* de la cerca, com s'apreciava als diagrames de seqüència UML.

Finalment, per mantenir l'execució de l'algorisme d'A*, s'utilitza la classe *Astar*, amb els corresponents contenidors, en forma de *maps*, per emmagatzemar quins nodes estan oberts i quins tancats (*openVertices* i *closedVertices*), i per emmagatzemar les distàncies dels nodes trobats fins al moment (*gScore*, de *s* fins al node, i *fScore*, *gScore* més la distància heurística des del node fins a *t*). A més de les funcions ja explicades per implementar el *heap* (per mantenir *openVertices* ordenat pel valor de *fScore*), s'implementa la funció de càlcul del valor heurístic des d'un node fins a un altre; l'obtenció d'aquest valor heurístic es comentarà més endavant. Cadascuna de les sub-classes – *AstarGen* i *AstarSpec* -, mantenen un *map* amb, per cada node del graf *G* (identificat amb el seu corresponent identificador), una llista de les arestes que arriben cap al node (en el cas d'*AstarSpec*, guardant l'identificador del node del qual arriba la relació, i en el cas d'*AstarGen*, guardant l'identificador del node més el tipus de transport, juntament amb el pes de l'aresta). També implementen el *main* de la cerca a A*, *GoAstar(int,bool)*.

Un cop explicada l'estructura de les classes darrera de la implementació de K*, podem passar a explicar com s'han implementat exactament els algorismes, definint la implementació de les funcions principals d'aquests, *GoKstar* i *GoAstar*. Per cadascuna de les funcions, s'explicaran els seus mòduls principals, i finalment, es donarà la complexitat de la funció i de l'algorisme en funció dels paràmetres d'entrada[17]. Primer s'explicarà la funció d'A* i després la de K*.

[17] Val a dir, però, que encara que es donaran les complexitats en funció de *n* (vèrtexs a la base de dades), *m* (arestes) i *k*, el paràmetre que realment escalarà al nostre sistema es *m*, ja que *n* representa el conjunt de ciutats, que serà, en general, constant.

Primer de tot, cal esmentar que l'algorisme d'A* té dues variants en relació al nombre d'iteracions de l'algorisme que s'efectuen: la primera només s'executa a la primera execució que es fa d'aquest, i es faran el nombre d'iteracions necessàries fins a trobar el node *t*; la segona, que s'executarà a la resta d'execucions, farà *2n* iteracions, on *n* és el nombre de vèrtexs trobats fins al moment al graf – és a dir, nodes tancats per A* -. A la Figura 21 es presenta un pseudocodi de la implementació d'A*.

Com es pot veure al pseudocodi, A* és l'encarregat de crear els *Hins* de *P(G)*, creant un *heap* per cada node que es tanqui a l'algorisme. A aquest *heap* s'afegiran, com a nodes de *P(G)*, tots els *sidretrack edges* que es conegui fins al moment que arriben al node en qüestió.

La resta de la lògica de l'algorisme és la típica a qualsevol A*, amb algunes variacions. Primer es tanca el node, i s'elimina de la llista d'oberts. En aquest moment es comprova si el node és *t*, i si efectivament ho és i s'està a la primera execució, es retorna *cert*. Sinó, a continuació s'itera cada node accessible des de l'actual (excepte el node *t* si l'actual és el node *s*, per evitar viatges directes[18]); aquí, cal puntualitzar, s'aplica un filtre dins de les *queries* a la base de dades si es treballa amb relacions Específiques, crucial per l'eficiència de l'algorisme, ja que, al treballar amb camins d'aquest tipus de relacions, només es tenen en compte les relacions que surtin del node destí de la relació representada pel node actual, quan tinguin una data de sortida dins del rang definit a les variables "*lookup*" de l'objecte *KstarSpec*. Així, s'evita afegir relacions a memòria que s'acabaran descartant posteriorment - explicat a continuació -, reduint també un gran nombre d'iteracions. Llavors, per cada node accessible, es realitzen comprovacions prèvies sobre aquest per descartar-lo en cas que no ens interessi; en el cas de treballar amb relacions Generals, només es comprova que no s'estiguin realitzant cicles; en el cas de les relacions Específiques, apart d'aquesta comprovació, es mira que el primer node (és a dir, la primera relació que surti de *s*) surti a la data indicada per l'usuari i, que pels altres nodes, la seva data de sortida sigui posterior a la d'arribada del node actual (més dues hores per transbordar) i no superior a un temps màxim d'espera per transbord (de dotze hores)[19]. Seguidament, si el següent node ja era tancat, hem de tenir en compte que quan es va tancar no es va tenir en compte la relació actual com a *sidetrack edge*, i per tant s'ha d'afegir al seu *Hin*, sempre i quan

[18] Realment això només es fa per les relacions Específiques, per les Generals, com només poden haver-hi tres viatges directes, simplement es comprova la longitud del viatge després de la passa de Dijkstra.
[19] Aquestes dues constants - les hores mínimes i màximes per transbordar -, també utilitzades a l'hora de passar les solucions a estat consistent, són subjectives i sempre susceptibles a canvis.

no interfereixi a la cerca de Dijkstra (línia 27). Això vol dir que, si no és la primera execució, ens hem d'assegurar que afegint el nou node al *Hin* no pertorbem la posició d'algun node que ja hagi estat obert per Dijkstra. A continuació el node tancat es descarta, no donant la possibilitat de reobrir-ho en cas que el nou camí sigui més curt, i per tant assumint la no optimalitat de l'algorisme amb heurístiques inconsistents – això s'explica amb més detall al següent apartat. Les següents operacions són les típiques per avaluar un node a A*, obtenint la seva distància al node origen, descartant-lo en cas que sigui superior a la coneguda, i si és inferior o no es coneix, actualitzant la informació corresponent i actualitzant la cua de prioritat.

En quant a la complexitat total de l'algorisme, aquesta depèn principalment en l'heurística utilitzada per guiar la cerca d'A*. A continuació es detalla l'heurística utilitzada i la seva repercussió a la complexitat temporal i espacial.

HEURÍSTICA PER A A*

La cerca d'A* és guiada per una funció d'avaluació *f(u) = g(u) + h(u)*, per a qualsevol vèrtex *u* del graf, on *g* és el cost conegut entre el node d'origen *s* i u, i *h* és l'estimació heurística del cost entre *u* i el vèrtex de destí *t*. Una heurística no optimista, és a dir, que doni un valor major a *c(u,t)* (el camí més curt entre *u* i *t* al graf), faria que l'algorisme donés solucions no òptimes, i a més, es revisitessin molts nodes; això mai és desitjable. D'altra banda, una heurística optimista o *admissible*, és aquella que compleix que *h(u) <= c(u,t)*, per a qualsevol vèrtex u. Encara més, es diu que una heurística és *consistent* quan per cada aresta *(u,v)* del graf es compleix que *h(u) <= c(u,v) + h(v)*. En general, si A* permet reobrir nodes tancats si la nova distància trobada és inferior a la mínima registrada, una heurística admissible assegura l'assoliment de solucions òptimes; tot i així, com s'ha dit, al nostre algorisme no permetem que es reobrin nodes, i per tant, en aquest sentit, no hi ha diferència entre una heurística admissible i una no admissible. En canvi, una heurística consistent ens proporciona els següents avantatges: primer, ens assegura l'optimalitat a la cerca; i segon, ens assegura que els nous nodes afegits a l'*Hin* d'un node ja tancat (línia 28) s'afegiran a l'última posició d'aquest, i per tant no caldrà comprovar l'estat dels altres nodes de l'*Hin* a Dijkstra.

A més, com s'ha explicat anteriorment, a K*, una heurística admissible ens assegura que l'estat de Dijkstra es mantindrà després d'una nova execució d'A*, i una heurística consistent ens assegura, a més, que no caldrà reconstruir el graf *P(G)*. En el primer cas, no ens importa gaire

que l'heurística sigui o no admissible, ja que no permetem mai que l'estat de Dijkstra canviï. I en el segon, com que, com s'explica a continuació, mai podrem estar segurs de la consistència de la nostra heurística, decidirem reconstruir sempre *P(G)* per cada nova execució d'A*.

En resum, s'utilitzi l'heurística que s'utilitzi, si aquesta no és consistent no s'estarà assegurant l'optimalitat de la cerca d'A*. Però, de la mateixa manera, no s'estaran reobrin nodes tancats, assegurant així que la complexitat de l'algorisme no arribarà a ser mai exponencial en el nombre de nodes. D'altra banda, i al nostre parer, a K* no és tan important que l'algorisme d'A* enregistri el camí òptim exacte entre *s* i *t*, ja que, encara que s'enregistri un de no òptim, l'òptim segueix sent trobable per K* mitjançant la cerca de Dijkstra sobre *P(G)*.

Donat el nostre problema, una heurística no es pot basar en res més que en informació prèvia que es tingui sobre els preus d'una ciutat a una altra. Concretament, s'ha decidit emmagatzemar informació sobre el cost mínim entre cada parella de ciutats per cada tipus de relació. És a dir, es mantenen dos *maps* de manera consistent a memòria (diguem, *costsGeneral* i *costsSpecific*), on per cada parella de ciutats *a*, *b*: s'enregistra el cost mínim trobat amb camins de relacions Generals entre *a* i *b* a *costsGeneral[a][b]*, i entre *b* i *a* a *costsGeneral[b][a]*; i el cost mínim trobat amb camins de relacions Específiques, amb *costsSpecific[a][b]* i *costsSpecific[b][a]*. Llavors, el valor heurístic retornat per arribar d'una ciutat a una altra, és a dir, el valor retornat per *getHeuristicValue(a,b)* (línia 36), serà la meitat del valor mínim dins del *map* corresponent. Si no existeix encara cap valor dins el *map* (és a dir, encara no s'han trobat camins entre les dues ciutats a execucions anteriors), es retorna 400, que equival al preu mínim per a una combinació A→Buenos Aires→B (per a A i B qualssevol). L'heurística presentada segurament es comporti de manera admissible i consistent la majoria de les execucions, ja que és gairebé impossible que es trobi un camí el doble d'econòmic que el més econòmic trobat fins aleshores (admissibilitat) i perquè hi ha una clara relació entre *c(u,v)* i *h(u)* i *h(v)* (consistència). Tot i així, en cap cas podem assegurar al 100% que l'heurística es comportarà sempre de manera admissible, i menys de manera consistent. En el pitjor dels casos – si l'heurística no és consistent -, com s'ha dit, A* no trobarà l'arbre de cerca òptim al graf i a més es podrien afegir nous nodes als *Hin* a posicions intermèdies d'aquests, havent de comprovar l'estat a Dijkstra dels nodes que quedin per sota.

La complexitat temporal d'A*, tenint en compte totes les execucions de l'algorisme, ve donada per: el nombre de vegades que s'executarà el bucle principal, que serà igual o menor al nombre de vèrtexs al graf, *n*; el nombre de vegades que s'executarà el bucle més intern (com a molt *m*, les arestes al graf) multiplicat pel cost d'afegir un nou node a la cua de prioritat, en total *O(mlogn)*; i la creació de tots els *Hins*, que implicarà com a màxim, recórrer les *m* arestes del

graf i fer un *push* de cadascuna a un *Hin*, que, com a molt, tindrà *m* nodes (encara que a la pràctica seran molts menys), amb *O(mlogm)*. Això ens dóna una complexitat de *O(n + mlogm + mlogn)*. Evidentment, si l'heurística no es comporta de manera consistent, hauríem de tenir en compte el temps per comprovar que no s'està pertorbant la cerca a Dijkstra (línia 26), que, en el pitjor dels casos, sería *O(mlogm)* (que per cada aresta haguéssim de recórrer un *Hin* amb totes les arestes del graf).

```
1  def GoAstar(arr int, start bool) returns bool :
2
3      i <- 0
4      n <- closedVertices.length * 2
5
6      while (not openVertices.empty) :
7
8          if (not start) :
9              if (i == n) return true
10
11
12         current <- openVerticesPQ.top
13         Crea Hin per a current, afegint cada sidetrack edge que li arribi.
14
15         closedVertices[current] <- true
16         openVertices[current] <- false
17         i++
18         if (start AND current == arr) return true
19
20         rels <- database.getOutgoingRels(current)
21         for (rel in rels) :
22
23             nextNode <- rel.end
24             Realitza comprovacions prèvies sobre nextNode, i descarta'l si és necessari.
25
26             if (closedVertices[nextNode]) :
27                 If (No es pertorba Dijkstra) :
28                     Afegeix rel com un nou node al Hin de nextNode.
29             Descarta el node.
30
31             tentativeGscore <- gScore[current] + rel.weight
32             if (tentativeGscore >= gScore[nextNode]) :
33                 Descarta el node.
34
35             gScore[nextNode] <- tentativeGscore
36             fScore[nextNode] <- tentativeGscore + getHeuristicValue(nextNode, t)
37             nextNode.CameFrom <- current
38
39             if (not openVertices[nextNode]) :
40                 openVertices[nextNode] <- true
41                 openVerticesPQ.push(nextNode)
42             else :
43                 openVerticesPQ.update(nextNode)
44
45
46     return false
47
48 end
```

Figura 21. Pseudocodi per a A*.

IMPLEMENTACIÓ DE K*

A la Figura 22 es presenta el pseudocodi per a la funció *GoKstar*. Aquesta té una estructura idèntica al flux d'execució de l'algorisme - definit anteriorment - i fa crides a les funcions *dijkstraStep()*, per portar a terme una passa de l'algorisme de Dijkstra, i *resumeAstar()*, quan calgui reprendre la cerca a A*; els pseudocodis per a aquestes funcions es presenten a les Figures 23 i 24 respectivament. La funció *schedulingMechanismEnabled()* simplement comprova que la cua de prioritat d'A* no estigui buida.

La funció *dijkstraStep()* fa les funcions comuns d'una passa de l'algorisme de Dijkstra: obté el node obert amb la menor distància, el tanca, i obre cadascun dels seus fills[20], afegint-los a la cua de prioritat amb una distància igual a la seva més Δ(*next*,*s*) i marcant-los com a utilitzats per Dijkstra. Finalment, es calcula el camí al graf original que suposa el node tancat i s'afegeix a la llista total de camins. Al conjunt de l'algorisme s'estaran executant *k* passes de Dijkstra, on a cadascuna es farà un *push* a la cua de prioritat, amb complexitat *O(klogk)*.

En quant a *resumeAstar()*, reprèn l'execució d'A* i seguidament reconstrueix el graf *P(G)*. A continuació s'han de repassar tots els nodes tancats a *P(G)*, ja que podria ser que, després d'aquesta nova execució d'A*, algun d'ells hagués obtingut un nou fill al graf (és a dir, a A* hagués trobat un nou *sidetrack edge* i l'hagués afegit al seu *Hin*), degut a que l'heurística no és 100% consistent; aquest fill s'hauria d'afegir a la cua de prioritat de Dijkstra. Una reconstrucció de *P(G)* implica recórrer com a molt *n* nodes i, per cadascun, afegir un nou node (l'arrel de l'*Hin*) a l'*Ht*; això comporta una complexitat de *O(nlogm*). Com a molt existiran *k* nodes tancats, per tant, la segona part de la funció tindria un cost de *O(klogk*). En total, una execució de *resumeAstar()* tindrà un cost de *O(nlogm + klogk)*. Com, cada cop que s'executa A* es troben el doble de nodes que la vegada anterior (si l'heurística no repeteix nodes), la funció s'executarà *logn* vegades, deixant una repercussió total de *(logn(nlogm + klogk))*.

Tenint en compte l'execució d'A* i l'execució de Dijkstra, la complexitat total de K* és de *O(n + mlogm + mlogn + klogk + logn(nlogm + klogk))*. Encara que linearítmica, aquesta complexitat podria reduir-se utilitzant *heaps de Fibonacci*, que reduirien la complexitat logarítmica dels *pushs* a les cues de prioritat a complexitat constant, deixant una complexitat final de *O(n + m + k + klogn + nlogn)*, que, sabent que n > 10, tindríem *O(m + klogn + nlogn)*.

Evidentment, *n* i *m* són els nodes i les arestes, respectivament, del graf *G*. En el cas de la cerca de camins amb relacions Generals, aquests valors també equivalen als del graf d'entrada de la base de dades, però amb les relacions Específiques, com es va dir a la Modelització del Domini, *n* passa a ser igual a les relacions de la base de dades i *m* al quadrat d'aquestes. Això implica que la complexitat real de K* per aquest tipus de cerca és de $O(m + m^2logm^2 + m^2logm + klogk + logm(mlogm^2 + klogk))$, on *n* i *m* són els nodes i les relacions a la base de dades, respectivament; i en el cas d'utilitzar *heaps de Fibonacci*, $O(m^2 + klogm + mlogm)$. Així, la complexitat de l'algorisme passa a ser quadràtica en el nombre de relacions. Per limitar l'impacte temporal que

[20] Recordem que un node de *P(G)* té com a molt 5 successors: dos al *Hin*, dos al *Ht* i un *cross edge*.

té això, cal limitar el nombre de relacions que es recorren per cada node a un nombre constant que ens asseguri temps d'execució raonables.

Per calcular la complexitat espacial de l'algorisme, hem de tenir en compte l'espai que requereix A*, l'espai consumit per *P(G)* i l'espai que requereix Dijkstra. En el pitjor dels casos, A* tindrà una complexitat espacial de *O(n)*. A *P(G)*, tots els nodes dels *Hin* implicaran *O(m)*, i tots els nodes dels *Ht* implicaran $O(n^2)$, per tant tot el graf tindrà una complexitat de $O(m + n^2)$. I sabem que Dijkstra visitarà *O(k)* nodes. Així la complexitat espacial total de l'algorisme és de $O(m + n^2 + k)$, per a les relacions Generals, i de $O(m^2 + k)$ per a les relacions Específiques.

```
def GoAstar(dep int, arr int) returns [][]int :

        foundSolutions <- [][]
        succesful <- goAstar(arr, true)
        if (not succesful) return [][]

        openD.push(R)
        createHt(R)

        while (not astar.openVertices.empty OR not openD.empty) :
                if (schedulingMechanismEnabled()) :
                        if (not openD.empty) :
                                if (Es compleix condicó de scheduling) :
                                        dijkstraStep()
                                        if (foundSolutions.length == K) :
                                                return foundSolutions
                                else :
                                        resumeAstar(arr)
                        else :
                                resumeAstar(arr)
                else :
                        dijkstraStep()
                        if (foundSolutions.length == K) :
                                return foundSolutions

        return foundSolutions

end
```

Figura 22. Pseudocodi per a K*.

```
def dijkstraStep() :

        next <- openD.pop()
        closed[next] <- true

        succ <- append(succCrossEdges(next), succHeapEdges(next))
        for (s in succ) :
                s.usedInDijkstra <- true
                s.dist <- next.dist + getDelta(next, s)
                parents[s].push(next)
                explored[s] <- true
                openD.push(s)

        path <- getPath(getTetaSeq(next))
        foundSolutions.push(path)

end
```

Figura 23. Pseudocodi per a *dijkstraStep()*.

```
def resumeAstar(arr int) :

        goAstar(arr, false)
        Reconstrueix P(G)

        for (node in closed) :
                succ <- append(succCrossEdges(node), succHeapEdges(node))
                for (s in succ) :
                        if (closed[s] OR explored[s]) continue
                        if (schedulingMechanismEnabled() AND No es compleix condicó de scheduling) :
                                resumeAstar(arr)
                        else :
                                explored[s] <- true
                                parents[s].push(node)
                                s.dist <- node.dist + getDelta(node, s)
                                openD.push(s)

end
```

Figura 24. Pseudocodi per a *resumeAstar()*.

6.2.2 TRANSFORMACIÓ DE CAMINS AL GRAF A CAMINS REALS

Amb l'objectiu d'aconseguir transformar els *k* camins trobats per K* a un estat consistent amb dades de viatges reals, primer cal passar aquests camins a llistes de *structs* representant trajectes entre ciutats amb informació sobre la ciutat de sortida, la d'arribada, el tipus de transport i, només en el cas de les relacions Específiques, la data de sortida. Això té una complexitat de *O(km)*, on *m* són les arestes del graf *G* de l'algorisme. A continuació es du a terme el següent procés per acabar aconseguint les solucions finals.

Per cadascun dels camins, es crida al sub-mòdul d'*scraping* per cada relació per a que busqui a la pàgina web corresponent (la que indiqui el tipus de transport de la relació) els trajectes disponibles per al dia indicat per la relació, en el cas de relacions Específiques, o per al dia de sortida que indiqués l'usuari i el següent[21], en el cas de les Generals. Un cop es tenen, per un camí, tots els trajectes reals, es cerca, per cada trajecte que surti de l'origen, d'entre totes les combinacions possibles, la que arribi abans a la ciutat de destí i la més econòmica. Això ho aconseguim mitjançant l'ordenació per preu i per hora d'arribada de cada conjunt de trajectes reals de cada relació del camí (excepte els de la primera), i recorrent cada conjunt per enllaçar els trajectes que siguin compatibles. Es veurà més fàcilment amb un exemple; suposem que, a partir del graf de la Figura 2, obtenim, entre d'altres solucions, la solució A-C-B, i, mitjançant l'*scraper* d'Aerolíneas Argentinas (suposarem que les relacions A-C i C-B eren ambdues d'aquest tipus), obtenim els resultats de la Taula 2 per a cada relació.

A - C	C - B
10:00-11:00 1.000$	13:00-15:00 800$
08:00-10:00 1.250$	06:00-08:00 1000$
16:00-17:00 1.000$	15:00-16:00 1.500$
	13:00-15:00 900$

Taula 2. Trajectes trobats per a A-C-B sense ordenar.

A continuació, s'ordenen els trajectes tant per preu com per hora d'arribada, excepte els de la primera relació (Taula 3 i 4, respectivament).

[21] Això ho fem per poder tenir transbords un dia després de la sortida. Realment, es podria fer que també tingués en compte dos, o fins i tot tres dies després.

A - C	C - B
10:00-11:00 1.000$	13:00-15:00 800$
08:00-10:00 1.250$	13:00-15:00 900$
16:00-17:00 1.000$	06:00-08:00 1000$
	15:00-16:00 1.500$

Taula 3. Trajectes trobats per a A-C-B ordenats per preu, amb les combinacions més econòmiques.

A - C	C - B
08:00-10:00 1.250$	06:00-08:00 1000$
10:00-11:00 1.000$	13:00-15:00 800$
16:00-17:00 1.000$	13:00-15:00 900$
	15:00-16:00 1.500$

Taula 4. Trajectes trobats per a A-C-B ordenats per hora d'arribada, amb les combinacions més ràpides.

D'aquesta manera es troba, per cada camí trobat per K*, les solucions més econòmiques (Taula 3) i les més ràpides (Taula 4) aprofitant tots els trajectes que surten el dia indicat per l'usuari. A la Figura 25 es presenta el flux d'execució del procés descrit anteriorment.

Primerament, s'executen, cadascuna en un *thread* diferent, les funcions de *scrape* corresponents (cridant a la funció *GetDayOffersAndRetain*) a cada relació de cada camí trobat per K*; la complexitat d'aquesta etapa de scrape és constant, *O(1)*, ja que cada relació compta amb un *thread* diferent i s'executen en paral·lel. Un cop es tenen aquests trajectes reals per a cada relació, s'ordena cada conjunt de cadascuna d'aquestes (excepte, com s'ha dit, el de la primera relació) tant per preu dels trajectes com per hora d'arribada, a dins del mateix *thread* creat abans per cadascuna de les relacions; tenint en compte que totes les ordenacions s'executen de manera paral·lela, aquesta operació té una complexitat de *O(tlogt)*, on *t* és el màxim nombre de trajectes trobats per una relació d'entre totes. D'aquesta manera, es genera, per cada camí, un bloc de trajectes ordenats per preu per cada relació, i un altre de trajectes

ordenats per hora d'arribada. Llavors, només cal recórrer, per cadascun dels blocs però en dos *threads* diferents, per cada trajecte del conjunt de la primera relació, tots els altres conjunts per buscar la primera combinació factible[22] a cadascun d'ells; el cost d'aquesta última etapa és, tenint en compte que ambdós blocs s'executen en paral·lel, de *O(pnr)*, on *p* és el nombre de trajectes trobats per a la primera relació, *n* és el màxim nombre de trajectes trobats per una relació d'entre totes, i *r* és el nombre de relacions al camí. D'aquesta manera s'obté, per cada camí i cada trajecte que surt de la ciutat d'origen, les combinacions més ràpides i més econòmiques.

SCRAPING

Només ens queda per comentar aquest sub-sub-mòdul que s'utilitza tant al sub-mòdul per trobar les combinacions reals pels camins de K* com als mòduls per trobar els viatges directes. L'implementa íntegrament la funció *GetDayOffersAndRetain* del paquet de *Scraping*. La funció, donada la informació de cerca del viatge (data de sortida, origen i destí, i número i tipus de passatgers) i el tipus de transport que correspongui, executa, primer, el mètode de cerca per obtenir la informació dels trajectes al servidor corresponent (*scrape*), amb complexitat constant, i després, el mètode per realitzar l'etapa de retenció de dades del sistema, actualitzant, per cada trajecte trobat, tant les relacions específiques com les generals, de manera paral·lela. El procés de retenció es realitza, en ambdós casos, amb una sola *query* a la base de dades per a tots els trajectes, que es triga en construir *O(t)*, on *t* són el nombre de trajectes trobats.

Cal puntualitzar una característica de la cerca d'informació als servidors (*scrape*), i és que retornarà sempre el preu més econòmic per un trajecte. És a dir, davant de la situació en que, per exemple, un vol tingués els preus per Turista o per Business, es retornaria el preu per Turista.

[22] Dos trajectes formen una combinació factible sempre que el segon surti almenys dues hores més tard que el primer i com a molt 12 hores més tard – igual que a l'etapa de cerca de camins a A*.

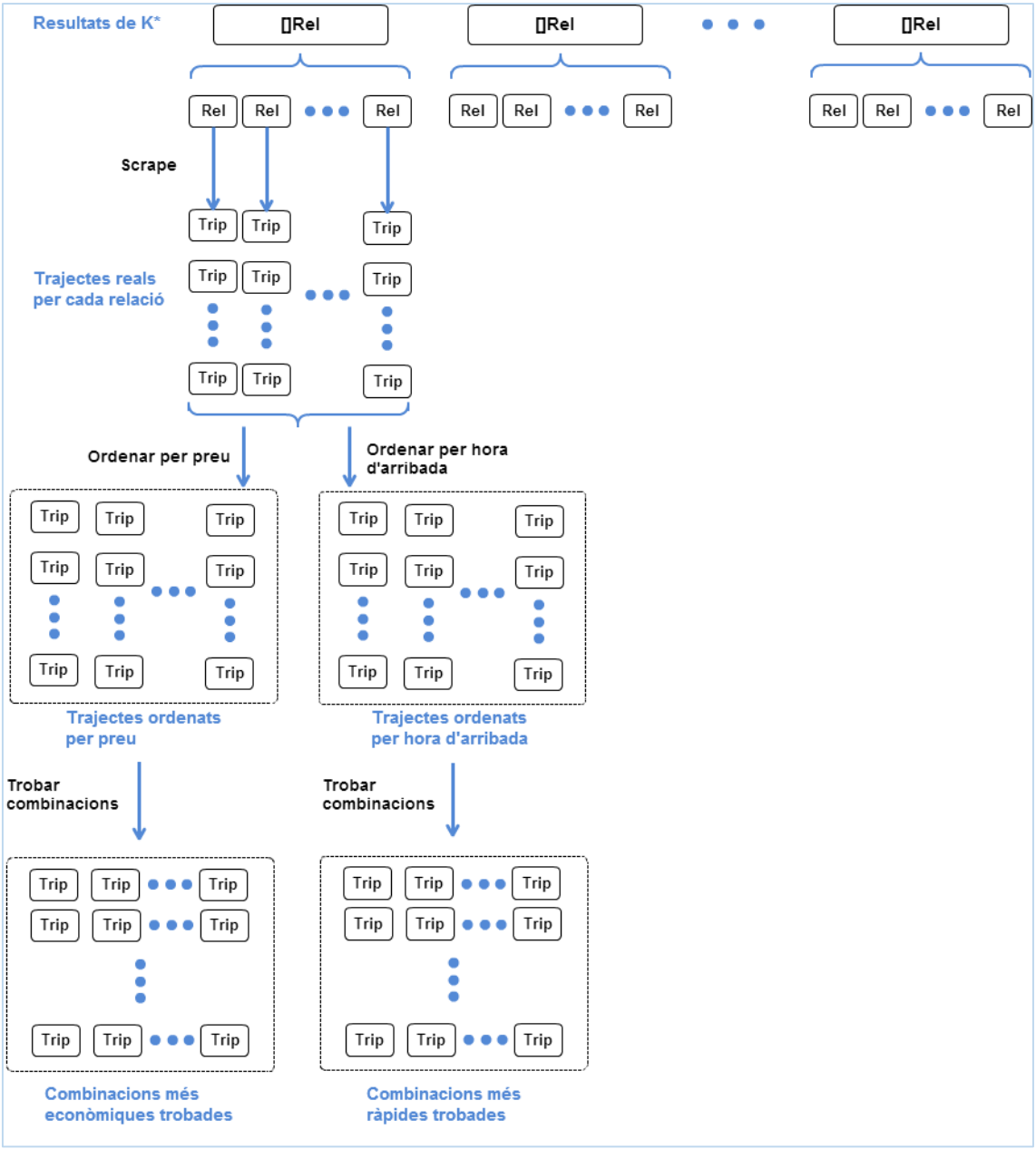

Figura 25. Flux d'execució per trobar les solucions finals.

6.2.3 FIXACIÓ DE CONSTANTS

Hi ha dues constants al sistema crucials per a l'eficiència i alhora l'eficàcia per obtenir viatges amb combinacions. La primera és el valor de *k*, és a dir, el màxim de solucions que buscarà i retornarà l'algorisme de K*, a cadascuna de les seves versions. L'altra és, com s'ha dit al explicar la complexitat temporal de K* amb relacions Específiques, el màxim nombre de relacions que es recorren per cada nou node que es tanca a A*, és a dir, el factor de ramificació d'A* per a relacions Específiques. A través de la pràctica, s'ha vist com uns valors per les constants *k=15* i *factor de ramificació=100*, donen bons resultats amb temps d'execució raonables, i són els que s'estan utilitzant ara per ara. A l'apartat 7.2.1 s'estudiarà l'efecte d'altres valors per aquestes dues constants i es justificarà la seva utilització.

7. AVALUACIÓ DEL SISTEMA

Aquest apartat té com a objectiu certificar la correctesa i l'efectivitat del sistema mitjançant l'execució i l'anàlisi de proves específiques. Primer de tot, per certificar la correctesa total del sistema, s'haurà de verificar cadascun dels mòduls identificats a l'apartat d'Implementació realitzant jocs de proves específics per a ells (tests unitaris), per finalment realitzar uns jocs de proves globals per a tot el sistema. Després, per convèncer-nos de l'efectivitat del sistema, aquest es sotmetrà a un aprenentatge constant durant un temps determinat, i s'anirà veient l'evolució de la qualitat de les solucions a mesura que el sistema emmagatzemi més coneixement.

7.1 AVALUACIÓ DE COMPONENTS

A continuació es passarà a demostrar la correctesa de cadascun dels mòduls definits anteriorment a l'etapa d'Implementació. Aquests mòduls són: el càlcul dels *k* camins amb K* i la transformació dels camins a un estat consistent. El primer mòdul engloba al sub-mòdul de la cerca al graf original amb A*, i el segon als sub-mòduls de *scraping* i de cerca de combinacions de trajectes. Primer de tot es provarà la correctesa del mòdul de K*, amb un apartat pel funcionament d'A* i un altre pel funcionament global de l'algorisme, i després es provarà el funcionament del segon mòdul, amb un apartat pel sub-mòdul de *scraping* i un altre pel funcionament global.

7.1.1 TESTS SOBRE K*

TESTS SOBRE A*

Seguidament es presenten un seguit de jocs de prova per comprovar la correctesa de l'algorisme d'A*. L'heurística, com que depèn completament de valors dependents de l'aprenentatge del sistema (el valor del camí més curt trobat fins al moment), s'ignorarà en aquestes proves, i prendrà valor 0; és a dir, A* actuarà com un Dijkstra. El comportament de l'heurística s'analitzarà al test d'aprenentatge del sistema, on podrem comprovar la seva admissibilitat. Primer presentem dos jocs de prova per comprovar la correctesa de la cerca amb relacions Generals i després dos jocs més per a la cerca amb relacions Específiques.

Primerament executem un test simple amb el graf de la Figura 26 com a estat de la BD, només amb relacions de tipus General – els números subratllats indiquen el tipus de transport de la relació i els altres el preu -, demanant a A* que calculi els camins entre els nodes 2 i 5, només executant-se fins a trobar el node de destí (modalitat de la primera execució a K*). L'algorisme acaba amb l'arbre de cerca representat a la Figura 27, i genera els *Hins* mostrats a la Figura 28. Amb aquest test es comprova que efectivament l'algorisme calcula el camí òptim entre el node d'origen i tots els altres que troba i que, a més, es creen correctament els *Hins* per a cadascun dels nodes tancats.

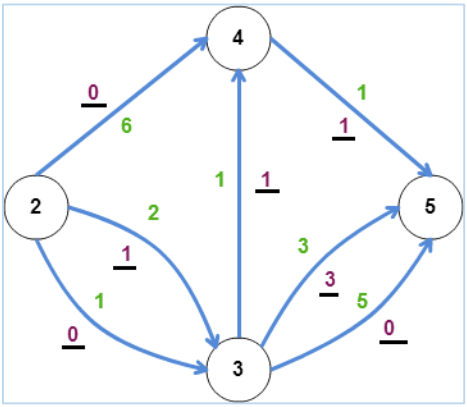

Figura 26. Test 1 per A*. Els números subratllats indiquen el tipus de transports i els altres el preu de les relacions.

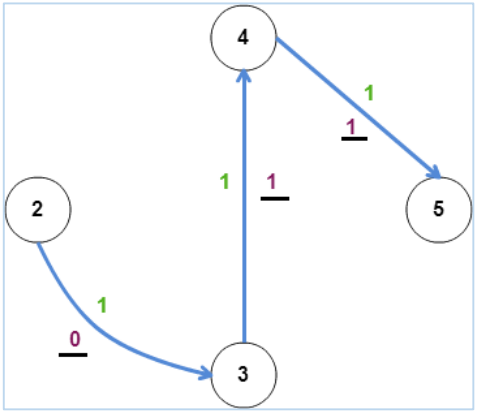

Figura 27. Resultats del test 1 per a A*.

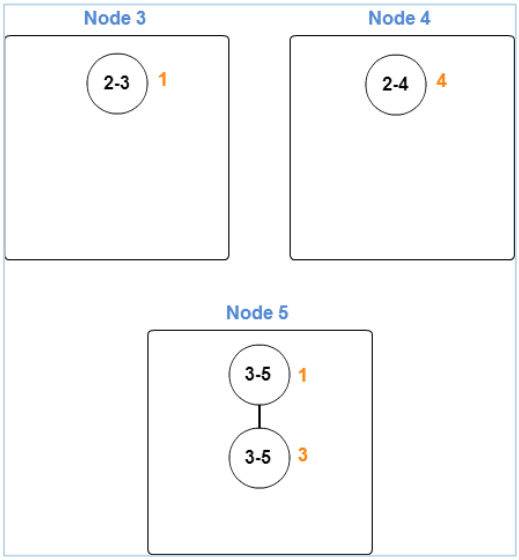

Figura 28. *Hins* creats al Test 1 per A – dValues al costat dels nodes.

Al següent test volem comprovar que l'algorisme, efectivament, descarta nodes quan es troba cicles, i que, després de la primera execució, tanca com a màxim el doble de nodes dels que havia tancat a l'execució anterior. Utilitzem com a graf d'entrada el mostrat a la Figura 29, i executem l'algorisme per a que trobi el camí més curt de 2 fins a 3. Quan l'algorisme acabi aquesta primera execució, seguirem executant-lo (en la segona modalitat) fins que la seva cua de prioritat sigui buida. A més, quan l'algorisme es trobi amb un cicle, li demanem que escrigui un missatge d'error.

Després de la primera execució, com s'espera, s'han tancat els nodes 2 i 3 i s'ha generat l'arbre de cerca de la Figura 30. A la segona, s'han tancat el doble de nodes i s'ha generat l'arbre de cerca de la Figura 31. Finalment, a la tercera, s'han tancat tots els nodes del graf i s'aconsegueix l'arbre de cerca final, representat a la Figura 32. A més, l'algorisme respon correctament amb els missatges d'error deguts als cicles entre 4 i 2 i entre 5 i 2.

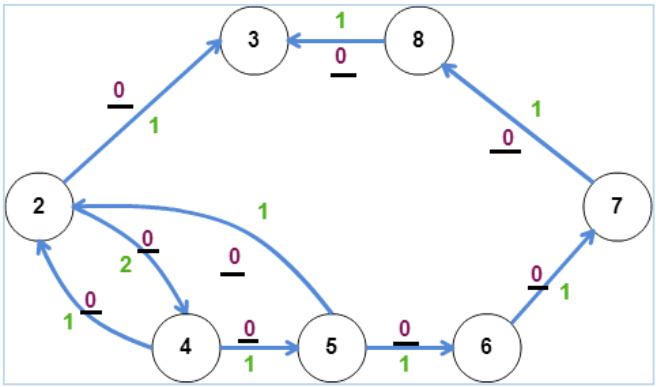

Figura 29. Test 2 per A*.

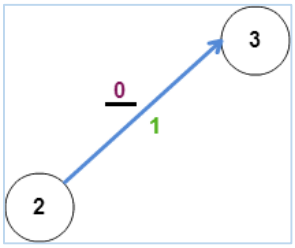

Figura 30. Primera execució Test 2 per A*.

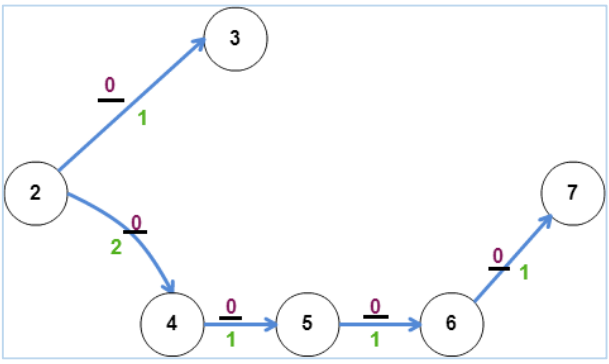

Figura 31. Segona execució Test 2 per A*.

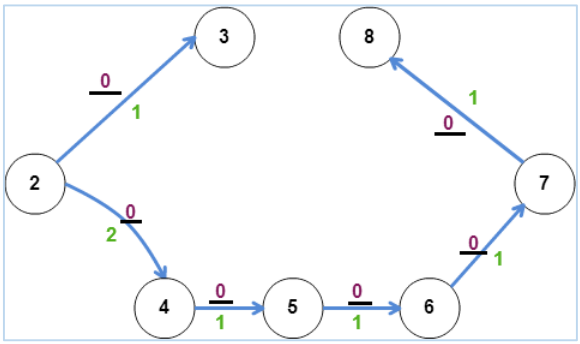

Figura 32. Tercera execució Test 2 per A*.

Per comprovar la correctesa d'A* amb relacions Específiques, es presentaran uns jocs de proves similars als dos anteriors, amb la diferència de que haurem de comprovar com l'algorisme descarta nodes si la seva data de sortida (recordem que els nodes representen relacions Específiques a la base de dades) no concorda amb la de l'anterior o si surten de la ciutat de sortida i la seva data de sortida no és igual a la introduïda per l'usuari. Com sabem, s'utilitza un graf alternatiu per a les relacions Específiques; tot i així, tant els grafs d'entrada com els resultats, excepte a l'últim joc, es presentaran en forma de graf de la base de dades, per facilitar la lectura.

Primer executem A* amb el graf de la Figura 33 a la base de dades (es mostra, per cada relació, el dia i l'hora de sortida i el preu) demanant el camí entre 2 i 5 pel dia 1. En ell, totes les relacions són accessibles per les altres, excepte la relació entre 2 i 6, que no s'hauria de tenir en compte ja que surt el dia 2. Correctament, l'algorisme construeix l'arbre de cerca mostrat a la Figura 34. A més a més, es comprova que els *Hins* es construeixen correctament després d'aquesta execució.

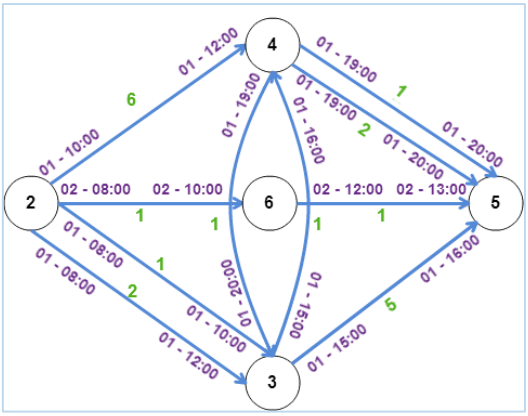

Figura 33. Test 3 per a A* - es mostra, per cada relació, el dia i hora de sortida i el preu.

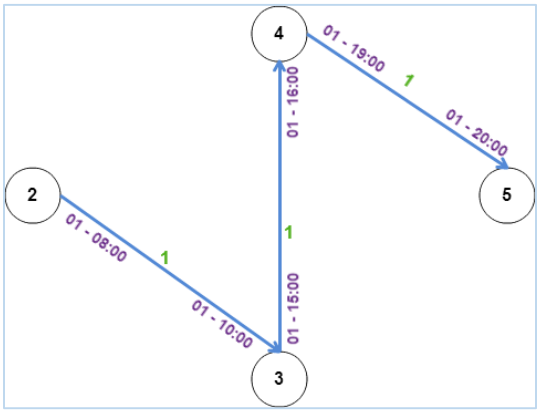

Figura 34. Execució Test 3 per a A*.

Per comprovar que no es tenen en compte relacions que no compleixin el mínim de dues hores per transbordar i el màxim de 12, es canvia l'hora d'arribada de la relació entre 2 i 3 de les 10:00 a les 14:00, i l'hora i el dia de sortida de la relació entre 4 i 5 al dia 2 a les 05:00. A més, s'afegeix una relació entre 4 i 3 per comprovar que l'algorisme detecta els cicles. El nou graf d'entrada es mostra a la Figura 35, subratllant els canvis introduïts, i el nou arbre de cerca trobat a la Figura 36. L'algorisme, correctament, agafa les relacions adequades entre 2 i 3 i entre 4 i 5, i mostra el missatge d'error pel cicle afegit.

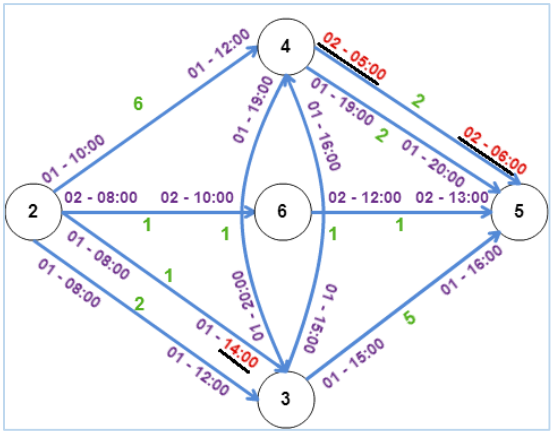

Figura 35. Test 3B per A*.

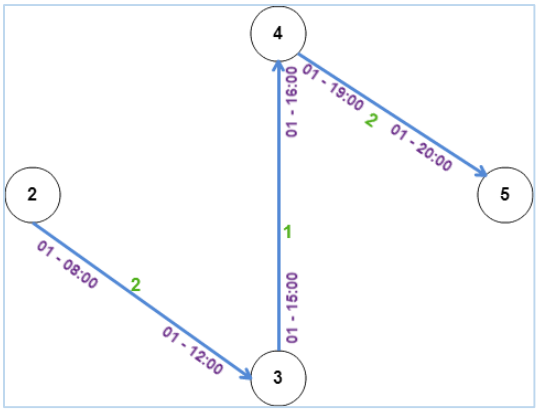

Figura 36. Resultat Test 3B per A*.

Finalment, per provar que a l'algorisme per relacions Específiques reprèn la cerca fins a trobar el doble de nodes que fins al moment, executem l'algorisme amb el graf de la Figura 37 (on totes les relacions són compatibles) per trobar els camins des del node 2 fins al node 3. A la Figura 38 es mostra el graf alternatiu que utilitza l'algorisme, tal i com es va explicar detalladament a l'apartat 4. L'algorisme, correctament, troba primer el camí més curt entre *a* i *b* a la primera execució (Figura 39); després, a la segona execució, tanca els 8 nodes següents més propers (el doble dels tancats anteriorment) (Figura 40); i finalment, a la tercera execució, tanca l'últim node que queda, el més llunyà (Figura 41).

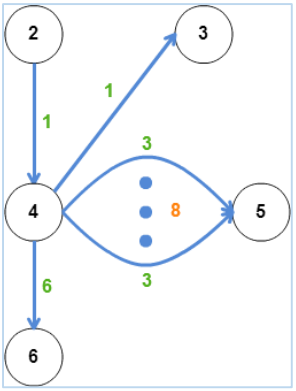

Figura 37. Test4 per A*.

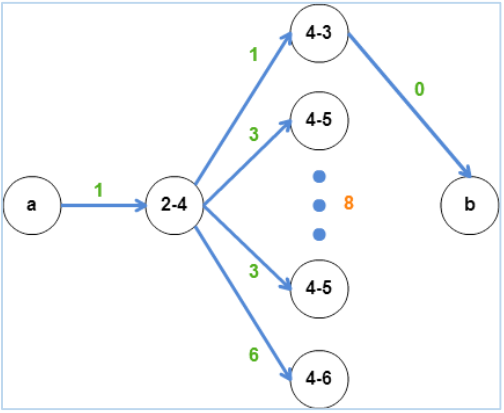

Figura 38. Graf alternatiu Test 4 per A*.

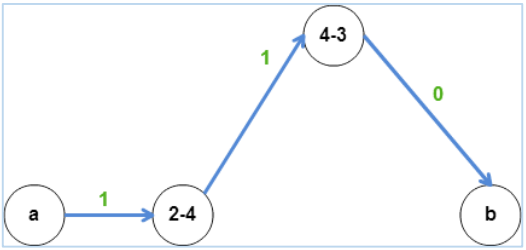

Figura 39. Primera execució Test 4 per A*.

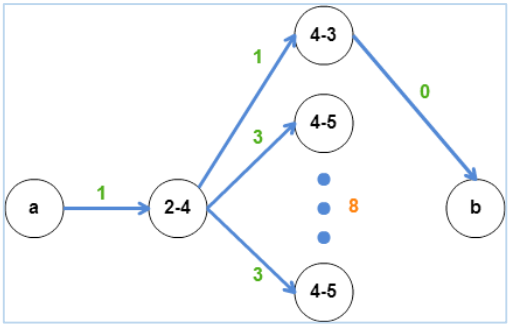

Figura 40. Segona execució Test 4 per A*.

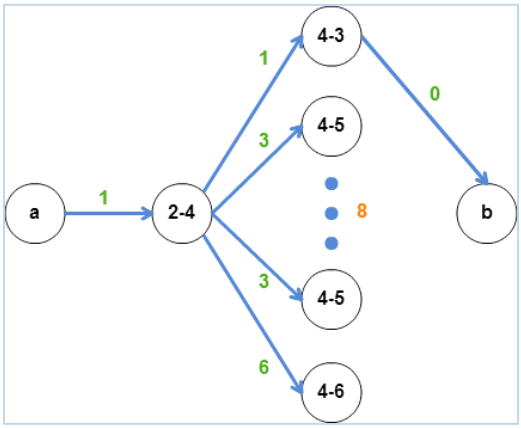

Figura 41. Tercera execució Test 4 per A*.

TESTS GLOBALS SOBRE K*

Un cop vist el correcte funcionament d'A*, es presenten els jocs de prova per comprovar la correctesa de K*. Primer es comprovarà el funcionament de l'algorisme sobre les relacions Generals, i després sobre les relacions Específiques. Per ambdues modalitats es mostraran gràficament els *k* camins més curts trobats, ordenats pel seu preu total, i només per a les relacions Generals s'il·lustrarà el graf *P(G)* present al sistema després de l'execució de l'algorisme – per a les Específiques, per simplicitat, no s'il·lustrarà, però sí que es comprovarà la seva correctesa.

Per comprovar l'algorisme sobre les relacions Generals, poblem de nou la base de dades amb el graf de la Figura 2, i executem l'algorisme per trobar els 4 camins més curts entre 2 i 5. L'algorisme respon correctament amb els 4 camins més curts i en l'ordre correcte; es representen a la Figura 42. A més a més, veiem com l'algorisme ha anat generant correctament el graf *P(G)* fins a assolir el seu estat final, il·lustrat a la Figura 43.

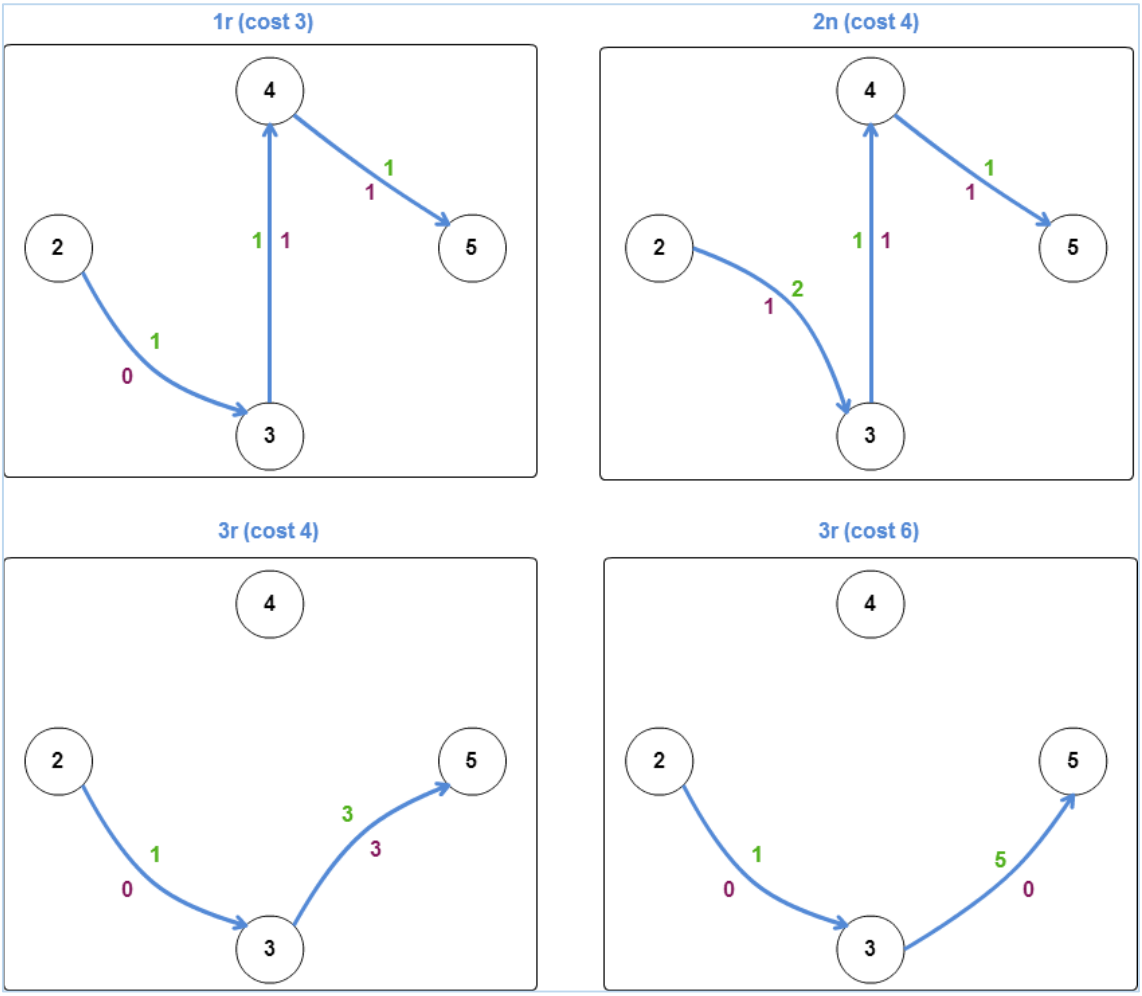

Figura 42. Execució Test 1 per a K*. K camins.

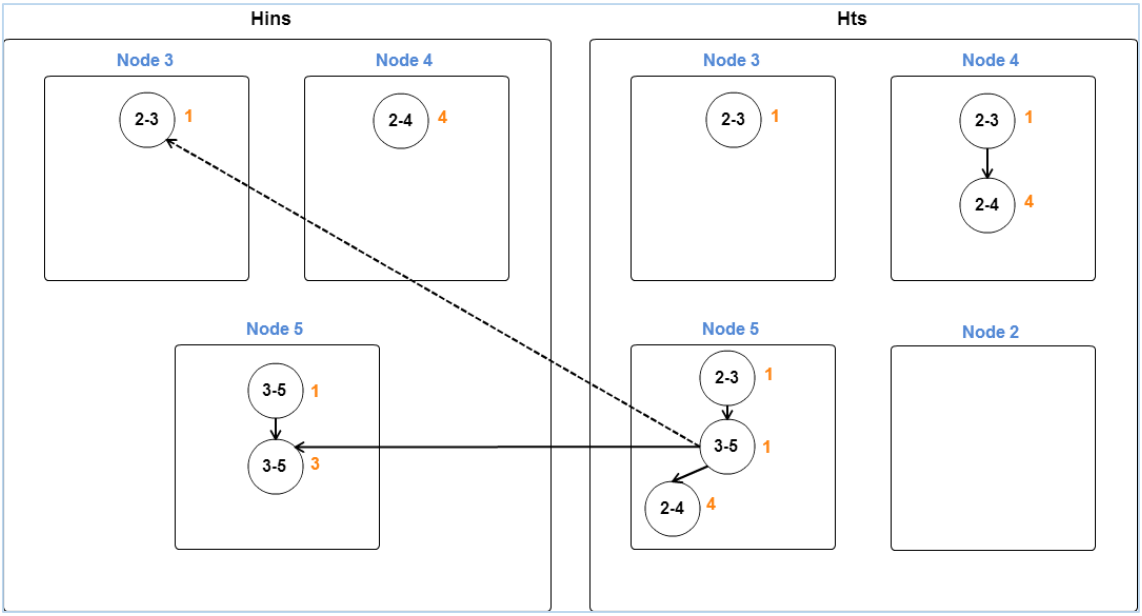

Figura 43. Execució Test 1 per a K*. *P(G)* – dValues al costat dels nodes.

Seguidament, passarem a provar el funcionament de l'algorisme K* sobre les relacions Específiques. Amb aquesta finalitat, tornem a poblar la base de dades amb el graf de la Figura 33, i demanem a K* que trobi les 6 combinacions més econòmiques entre 2 i 5. L'algorisme, correctament, troba els 6 camins més curts, mostrats a la Figura 44. Encara que, com s'ha dit, per simplicitat no s'inclou la representació de *P(G)*, es comprova que el graf ha estat construït correctament després de l'execució.

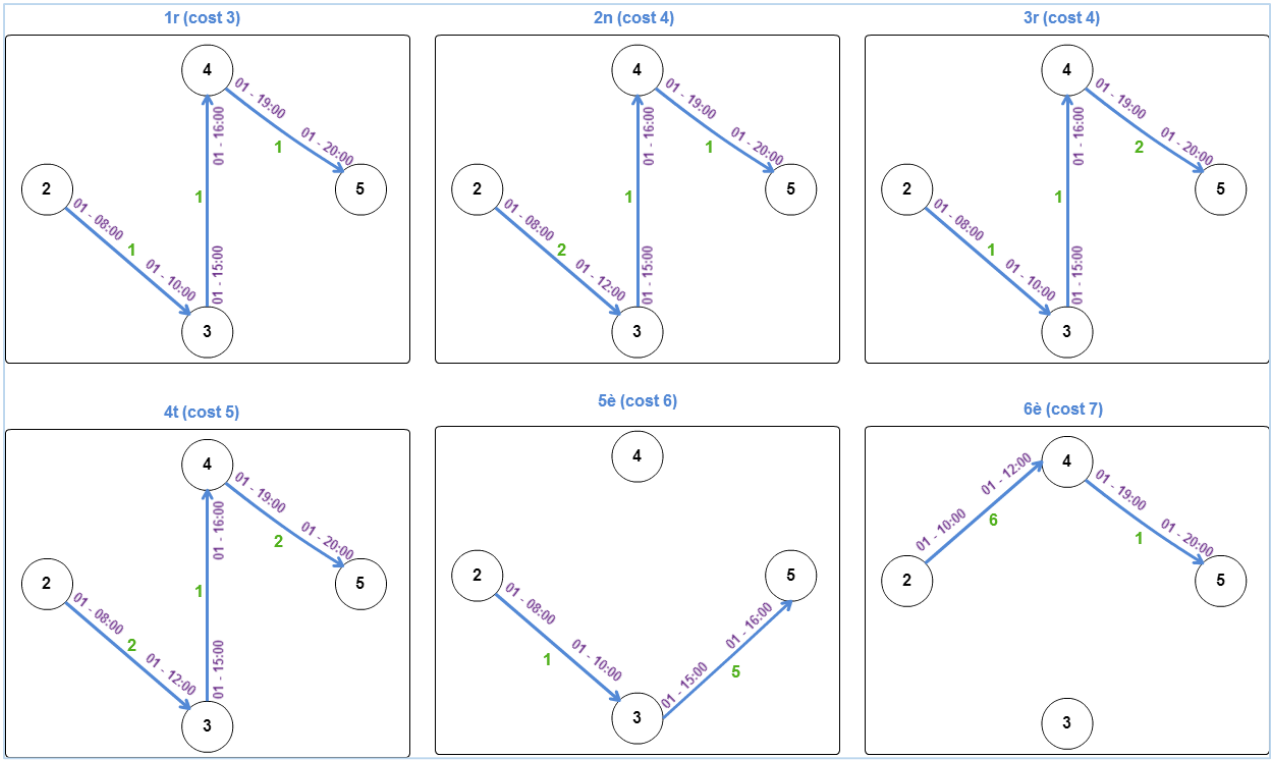

Figura 44. Execució Test 2 per a K*. K camins.

7.1.2 TESTS SOBRE LA TRANSFORMACIÓ DE CAMINS

Un cop realitzats els tests per assegurar el correcte funcionament del mòdul de la cerca dels *k* camins òptims, passarem a comprovar el segon mòdul encarregat de la transformació dels camins a un estat consistent. Dividirem la tasca en dues: realitzar tests sobre les funcions de *scraping* i d'actualització de la base de dades (*aprenentatge*); i realitzar tests sobre l'algorisme per obtenir les combinacions, que com sabem utilitza el sub-mòdul anterior.

Per comprovar el correcte funcionament d'aquestes dues funcionalitats, cridarem a la funció *GetDayOffersAndRetain* amb una entrada arbitrària i verificarem que es mostren els resultats que podem trobar a les respectives pàgines web i que a més la base de dades s'actualitza de manera correcta. Concretament, comencem l'execució amb una base de dades buida, i executem la funció demanant-li que busqui els trajectes disponibles entre Ciudad de Buenos Aires i Ushuaia pel dia 14 de juny de 2016, per a un adult amb Aerolíneas Argentinas. La funció respon amb els resultats de la Taula 5 (s'ignoren varis atributs sense transcendència), que concorden amb els de la pàgina web en qüestió.

Viatge	Preu
BUE→USH	3019$
BUE→USH	3019$
BUE→USH	3395.2$
BUE→USH	3643.91$

Taula 5. Test sobre *GetDayOffersAndRetain*. Resultats *scrape* 1.

Després d'aquesta execució, la base de dades ha sigut poblada amb quatre relacions Específiques entre els dos nodes en qüestió amb la informació pertinent, i amb una relació General pel tipus de transport d'Aerolíneas Argentinas entre els dos nodes, amb el preu mig dels quatre trajectes (3269.2775$).

Seguidament, per veure que no només es crea, sinó que s'actualitza correctament la informació dins de la base de dades, tornem a executar *GetDayOffersAndRetain* entre Buenos Aires i Ushuaia però pel dia 22 de juliol. Obtenim la informació mostrada a la Taula 6.

Viatge	Preu
BUE→USH	1839.19$
BUE→USH	3019$
BUE→USH	3395.2$
BUE→USH	3643.91$

Taula 6. Test sobre *GetDayOffersAndRetain*. Resultats *scrape* 2.

Amb aquesta nova informació, la base de dades ha sigut poblada amb quatre relacions específiques més, i s'ha actualitzat el preu de la relació General esmentada anteriorment a 3121.80125$, que equival a la mitjana del conjunt de trajectes trobats entre ambdues cerques.

TESTS SOBRE L'ALGORISME PER TROBAR LES COMBINACIONS

Un cop hem comprovat el funcionament de *GetDayOffersAndRetain*, podem passar a verificar la funció de cerca de combinacions donat un conjunt de camins d'entrada (recordem que els camins estaven formats per un seguit de relacions, amb el tipus de transport, la ciutat de sortida i d'arribada i, en el cas de les Específiques, la data de sortida). Aquesta verificació es dividirà, com és habitual, en la verificació de la funció per a camins de relacions Generals i per a camins de relacions Específiques. A ambdues parts, cridarem a la funció en qüestió amb un camí d'entrada amb dues relacions (de l'estil *ciutat1→ciutat2→ciutat3*).

Per provar el funcionament de la funció sobre relacions Generals, la cridarem simulant una execució anterior de K* que ha retornat el següent camí de relacions Generals per arribar des de Buenos Aires fins a Salta (les relacions s'escriuen amb la forma *[idOrigen, idDestí, transport]*): ([BUE, JUJ, Aerolíneas],[JUJ, SLA, Bus]), on BUE, JUJ i SLA són les ids a la base de dades de Buenos Aires, Jujuy y Salta, respectivament (és a dir, cal anar amb avió a Jujuy desde Buenos Aires, i després agafar l'autobús fins a Salta). Se li passa a la funció, a més, el dia de sortida de l'usuari, que suposarem com a 8 de juny de 2016; la funció, com es va explicar a l'apartat

d'Implementació, hauria de buscar els trajectes per a les relacions tant pel dia 8 com pel dia 9 (excepte la relació de sortida), i llavors trobar les combinacions adequades.

Els trajectes entre Buenos Aires i Jujuy amb Aerolíneas Argentinas pel dia 8 es representen a la Taula 7, i els de Jujuy a Salta amb autobús, pels dies 8 i 9, a la Taula 8. Els resultats de la funció es mostren a la Taula 9. L'algorisme troba, correctament, les combinacions més ràpides i més econòmiques per cada trajecte de la primera relació; noti's, però, que tant la combinació més ràpida com la més econòmica són la mateixa per a cada trajecte de sortida, i per tant l'algorisme les fusiona en una de sola.

Viatge	Dia i hora	Preu
1 - BUE→JUJ	8 – 8:40→10:55	1988.22$
2 - BUE→JUJ	8 – 10:40→12:55	1988.22$
3 - BUE→JUJ	8 – 17:50→20:05	1988.22$

Taula 7. Test per trobar combinacions. BUE->JUJ.

Viatge	Dia i hora	Preu
1 - JUJ→SLA	8/9 – 01:20→03:30	140$
2 - JUJ→SLA	8/9 – 05:20→07:40	140$
3 - JUJ→SLA	8/9 – 07:00→09:00	140$
4 - JUJ→SLA	8/9 – 09:00→11:00	140$
5 - JUJ→SLA	8/9 – 09:35→11:30	175$
6 - JUJ→SLA	8/9 – 12:00→14:00	140$
7 - JUJ→SLA	8/9 – 14:00→16:00	140$
8 - JUJ→SLA	8/9 – 14:01→16:15	140$
9 - JUJ→SLA	8/9 – 15:30→17:30	140$

10 - JUJ→SLA	8/9 – 16:35→18:45	140$
11 - JUJ→SLA	8/9 – 17:45→20:30	175$
12 - JUJ→SLA	8/9 – 18:45→21:00	140$
13 - JUJ→SLA	8/9 – 20:00→22:00	140$
14 - JUJ→SLA	8/9 – 20:20→22:30	140$

Taula 8. Test per trobar combinacions. JUJ->SLA.

Combinació	Dia i hora	Preu
1→7	8 – 8:40→10:55 8 – 14:00→16:00	1988.22$ + 140$
2→9	8 – 10:40→12:55 8 – 15:30→17:30	1988.22$ + 140$
3->1(dia 9)	8 – 17:50→20:05 9 – 01:20→03:30	1988.22$ + 140$

Taula 9. Test per trobar combinacions. Combinacions trobades (Generals).

D'una manera similar provarem el funcionament de la funció per a relacions Específiques. Simularem una resposta de K* amb el següent camí de relacions (en aquest cas les relacions tenen el format *[idOrigen, idDestí, transport, dataSortida]*): ([BUE, TUC, Aerolíneas, 08/06/2016],[TUC, SLA, Bus, 09/06/2016]), on TUC és el id de Tucumán a la base de dades. Els trajectes per a cadascuna de les relacions es representen a les Taules 10 i 11 respectivament. A la Taula 12 es mostren les combinacions trobades per l'algorisme. Com pot veure's, els únics trajectes de la relació inicial que poden combinar-se complint la restricció de tenir 12 hores de transbord com a màxim, són 4 i 5; i, novament, tant les combinacions més econòmiques com les més ràpides són les mateixes per ambdós trajectes.

Viatge	Dia i hora	Preu
1 - BUE→TUC	8 – 07:15→09:10	2606$
2 - BUE→TUC	8 – 09:25→11:20	2606$
3 - BUE→TUC	8 – 14:35→16:30	2606$
4 - BUE→TUC	8 – 17:40→19:35	2606$
5 - BUE→TUC	8 – 20:50→22:45	2606$

Taula 10. Test per trobar combinacions. BUE->TUC.

Viatge	Dia i hora	Preu
1 - TUC→SLA	9 – 04:55→08:55	340$
2 - TUC→SLA	9 – 05:55→10:05	386$
3 - TUC→SLA	9 – 09:15→12:55	369$

Taula 11. Test per trobar combinacions. TUC->SLA.

Combinació	Dia i hora	Preu
4→1	8 – 17:40→19:35 9 – 04:55→08:55	2606$ + 340$
5→1	8 – 20:50→22:45 9 – 04:55→08:55	2606$ + 340$

Taula 12. Test per trobar combinacions. Combinacions trobades (Específiques).

7.1.3 TEST GLOBAL SOBRE EL SISTEMA

Amb la finalitat de verificar que el sistema funciona com s'esperava, és a dir, que és capaç d'enregistrar nova informació al graf per utilitzar-la més tard per cercar combinacions tant a partir de relacions Generals com d'Específiques, es realitzaran uns tests amb una base de dades buida que es poblarà inicialment a partir d'un seguit de peticions de viatges Directes entre quatre ciutats del graf (Ushuaia, Buenos Aires, Córdoba y Salta); concretament, per cada ciutat, es faran quatre peticions de viatges directes cap a cadascuna de les altres tres ciutats, des del 8 fins al 11 de juny, per cada tipus de transport (és a dir, 12 peticions per cada parella de ciutats). Un cop poblat el graf amb relacions entre aquestes quatre ciutats, cridarem a les peticions de cerca de combinacions amb relacions Generals i amb relacions Específiques per a que cerquin els viatges entre Ushuaia i Salta pel dia 8 de juny. Per simplicitat no es mostraran els resultats obtinguts.

Demanant les combinacions amb un valor de *k* igual a 6, el sistema retorna, per a les peticions de cerca amb relacions Generals, 12 resultats, i amb relacions Específiques, 7 resultats. Si augmentem el valor de *k* a 15, es retornen 32 resultats per a les relacions Generals, i 12 resultats per a les Específiques. Finalment, si augmentem el valor a 100, es troben 35 combinacions amb les relacions Generals, i 22 amb les Específiques.

7.2 AVALUACIÓ GLOBAL

Dins d'aquest apartat demostrarem el funcionament total de l'aplicació final. Primer de tot posarem a prova el seu funcionament intern, realitzant un test sobre el *back-end* amb l'objectiu de verificar la seva capacitat d'obtenir millors solucions amb el temps, a més de la seva eficiència. Després es realitzarà una prova de funcionament sobre la interfície i es compararan els resultats amb els de pàgines web similars.

7.2.1 TEST D'APRENENTATGE SOBRE EL *BACK-END*

De cara a avaluar l'aplicació de manera global i mesurar la seva eficàcia i eficiència, es realitzarà una prova d'aprenentatge intensiu sobre el sistema, amb la finalitat de comprovar que, efectivament, a mesura que aquest rep noves cerques d'usuaris i emmagatzema la informació obtinguda al graf de la base de dades, no només s'obtenen més solucions sinó que la qualitat d'aquestes també s'incrementa (entenent qualitat com el preu final del viatge). A més, comprovarem a la vegada, com es comporta la nostra heurística d'A* i quins temps d'execució té aquest algorisme a mesura que el graf es fa cada cop més dens.

Aquesta prova es basarà en triar cinc parelles de ciutats que donin marge a l'obtenció de viatges amb combinacions (és a dir, que estiguin força lluny l'una de l'altra) amb l'objectiu de veure com evolucionen els viatges entre cada parella per a un dia determinat *d*. Per assolir-ho, a partir d'una base de dades buida, s'alternarà contínuament entre una *etapa d'aprenentatge* i una altra *d'obtenció de solucions*. L'etapa d'aprenentatge estarà constituïda per l'enviament al servidor de 50 peticions paral·leles de viatges directes entre parelles de ciutats aleatòries, amb una data de sortida *d*, *d+1*, *d+2* o *d+3* (ja que és el rang de dates al qual buscarà K* per a relacions Específiques); després de rebre les respostes s'enregistrarà el nombre de relacions presents a la base de dades. Seguidament, a l'obtenció de solucions, s'enviaran, per cada parella de ciutats i en paral·lel, les dues peticions pet obtenir viatges amb combinacions (Generals i Específiques). De la unió d'aquestes dues respostes, enregistrarem la informació referent al preu del viatge més econòmic trobat i al nombre de viatges trobats; a més, per cadascuna de les dues cerques amb A* (dins de l'algorisme global de K*), guardarem si l'heurística ha resultat inconsistent (és a dir, si s'ha intentat expandir un node ja tancat amb un cost menor del que tenia enregistrat) i el temps total d'execució. A la Figura 45 es mostra el flux d'execució del test.

El test es realitzarà amb les constants comentades a l'apartat d'Implementació *k=15* i *factor de ramificació=100*. Quan acabi el test, aprofitarem l'estat dens del graf de la base de dades per comprovar l'efecte al sistema de valors més grans per a aquestes dues constants.

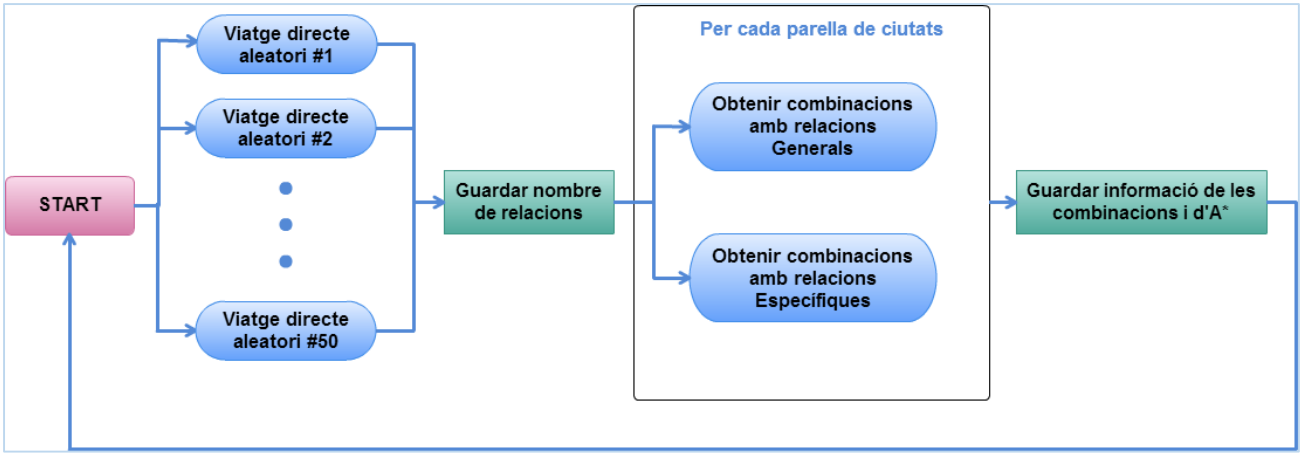

Figura 45. Flux d'execució del test d'aprenentatge global.

Es trien les 5 parelles de ciutats següents: *Ushuaia→Salta*, *Puerto Iguazú→Ushuaia, Jujuy→Calafate, Río Gallegos→Mar del Plata* i *Puerto Iguazú→Calafate*. El dia de sortida per a les combinacions serà el 10 de juny de 2016; és a dir, el 10, 11, 12 o 13 de juny per a les cerques directes de l'etapa d'aprenentatge.

Al Diagrama 1 es mostra l'evolució del nombre de relacions Generals a la base de dades, i al 2 la del nombre de relacions Específiques, fins a la iteració número 60. Com es pot apreciar, sembla ser que arribats a aquesta iteració, no queden molts més viatges directes per enregistrar a la base de dades que no hagin estat ja trobats. Així doncs, l'execució s'atura en aquest moment (després d'uns 90 minuts d'execució). Als Diagrames 3, 4, 5, 6 i 7, es mostra l'evolució dels preus trobats per a les 5 parelles de ciutats; es veu clarament com a totes elles s'han anat trobant preus més baixos a mesura que s'anava omplint la base de dades (amb l'excepció de *Uhusia→Salta*, que sembla ser que va perdre la disponibilitat d'algun dels trajectes claus per a realitzar les combinacions). Les variacions als preus cap a les últimes iteracions de la parella *Puerto Iguazú→Calafate* (Diagrama 7) poden ser degudes a problemes amb la connexió als servidors externs. Al Diagrama 8 es mostren, conjuntament, les línies de tendència de l'evolució del nombre de solucions trobades per cadascuna de les parelles al llarg de les iteracions; veiem com, excepte al cas de *Ushuaia→Salta*, totes tenen una tendència positiva i, en general, troben més solucions com més iteracions es realitzen.

En quant als temps d'execució de l'algorisme (K*, és a dir, sense tenir en compte el temps per a la cerca de combinacions reals), aquests han sigut de com a màxim 8 segons, en el cas de les relacions Específiques, i de menys de mig segon en el de les Generals – temps totalment acceptables. Als Diagrames 9 i 10 es mostra el creixement dels temps d'execució per a ambdós

tipus de relacions, traçant, per cada iteració, la mitjana del temps d'execució de les 5 parelles. Si comparem el creixement dels temps d'execució amb el del nombre de relacions al graf (Diagrames 1 i 2), es veu clarament com, en ambdós casos, el temps d'execució de K* creix linealment en relació al nombre d'arestes al graf.

Finalment, en quant a l'execució de l'algorisme d'A*, a les 600 execucions d'aquest portades a terme (60 iteracions x 5 parelles x 2 K* per parella), l'heurística s'ha comportat de manera consistent, així que queda clarament demostrada la robustesa d'aquesta.

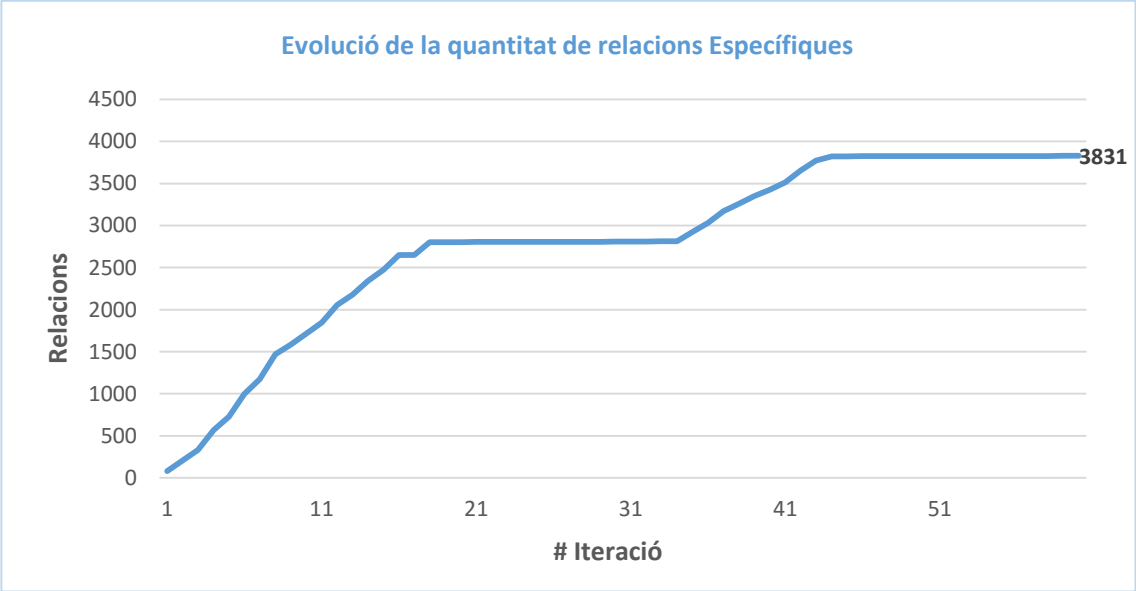

Diagrama 1. Test d'aprenentatge global. Evolució de la quantitat de relacions Específiques a la base de dades.

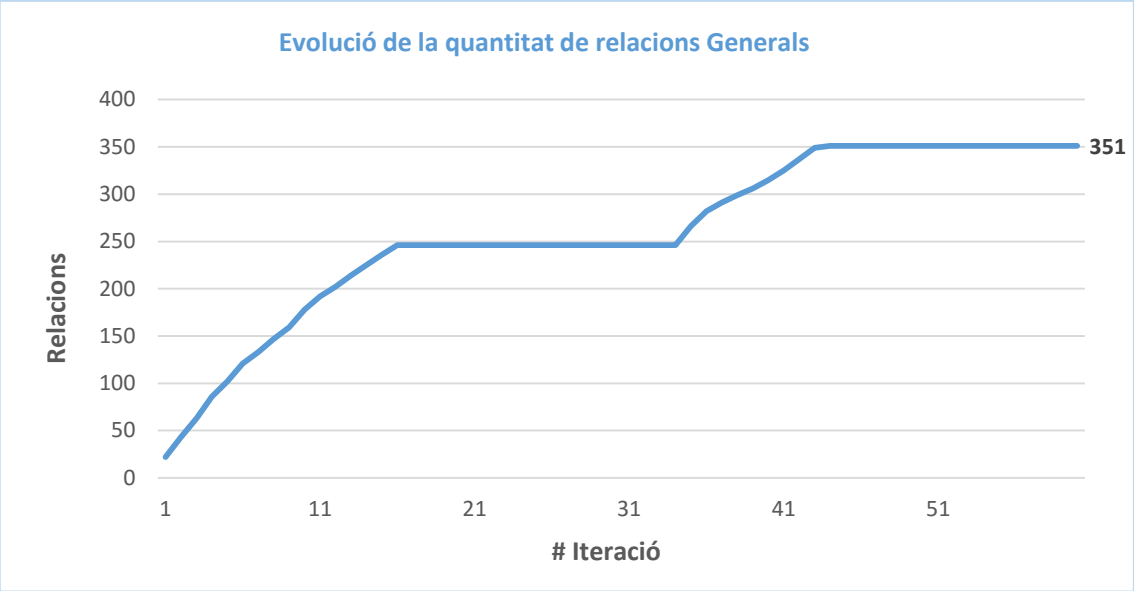

Diagrama 2. Test d'aprenentatge global. Evolució de la quantitat de relacions Generals a la base de dades.

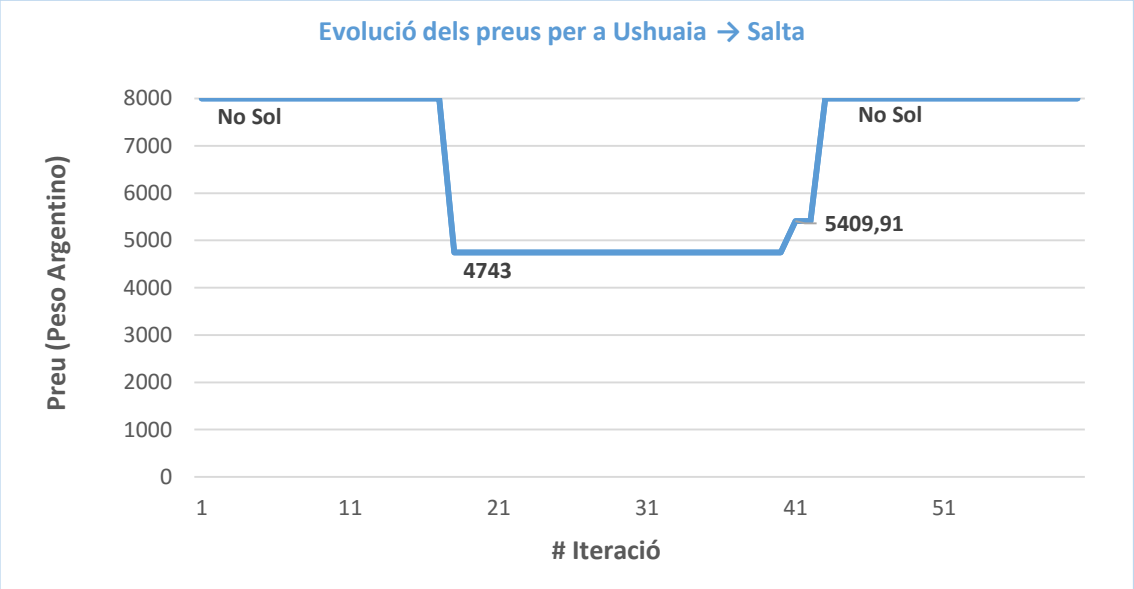

Diagrama 3. Test d'aprenentatge global. Evolució del preu per a Ushuaia->Salta – "No Sol" indica que no s'ha trobat cap combinació.

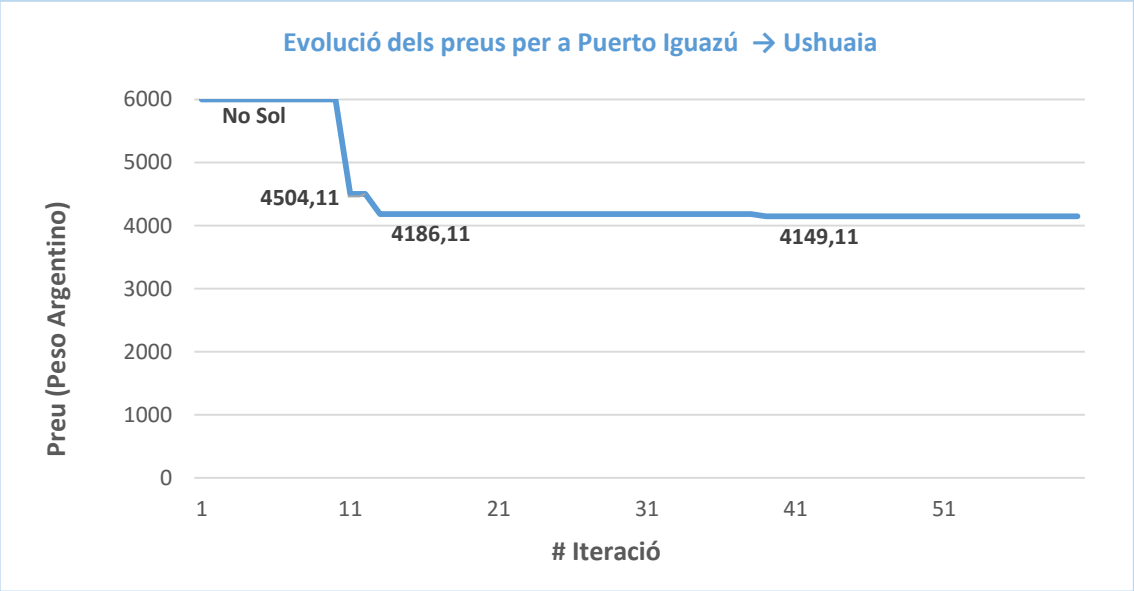

Diagrama 4. Test d'aprenentatge global. Evolució del preu per a Puerto Iguazú->Ushuaia – "No Sol" indica que no s'ha trobat cap combinació.

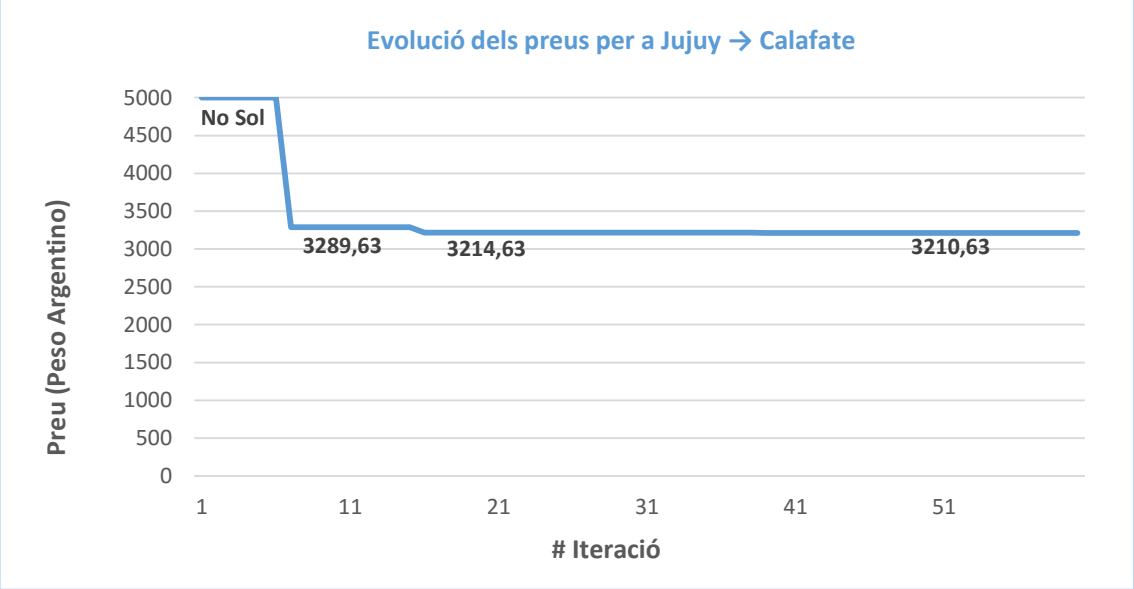

Diagrama 5. Test d'aprenentatge global. Evolució del preu per a Jujuy->Calafate – "No Sol" indica que no s'ha trobat cap combinació.

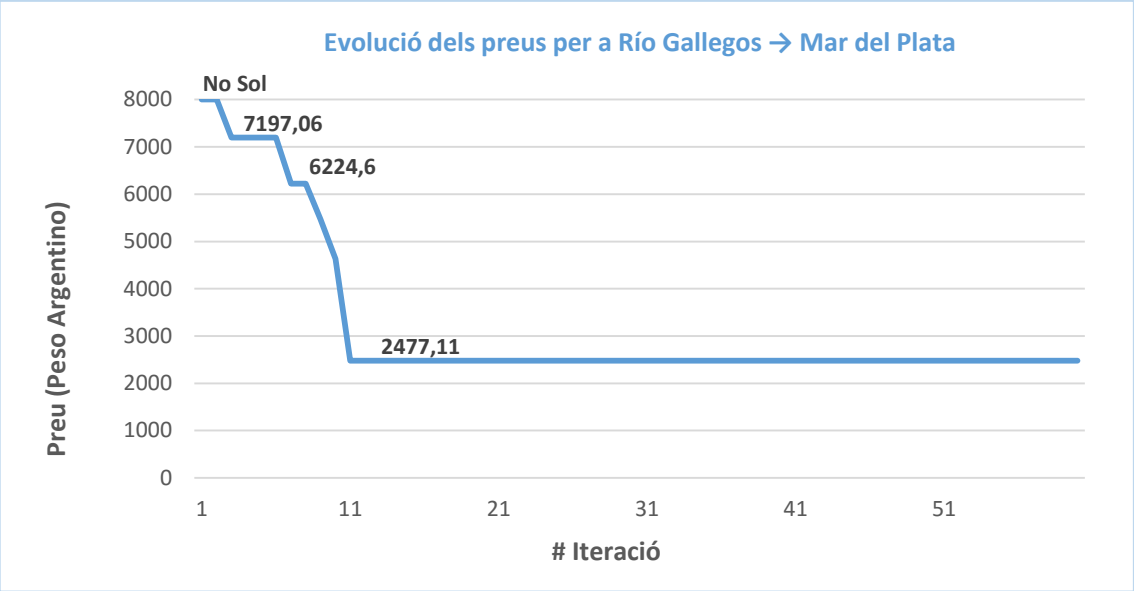

Diagrama 6. Test d'aprenentatge global. Evolució del preu per a Río Gallegos->Mar del Plata – "No Sol" indica que no s'ha trobat cap combinació.

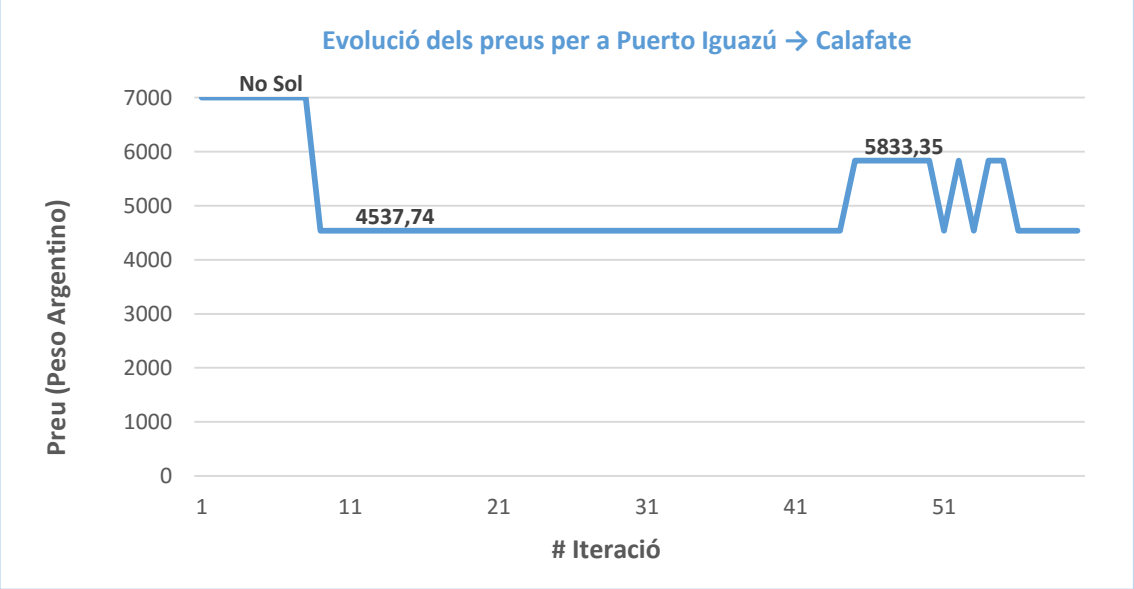

Diagrama 7. Test d'aprenentatge global. Evolució del preu per a Puerto Iguazú->Calafate – "No Sol" indica que no s'ha trobat cap combinació.

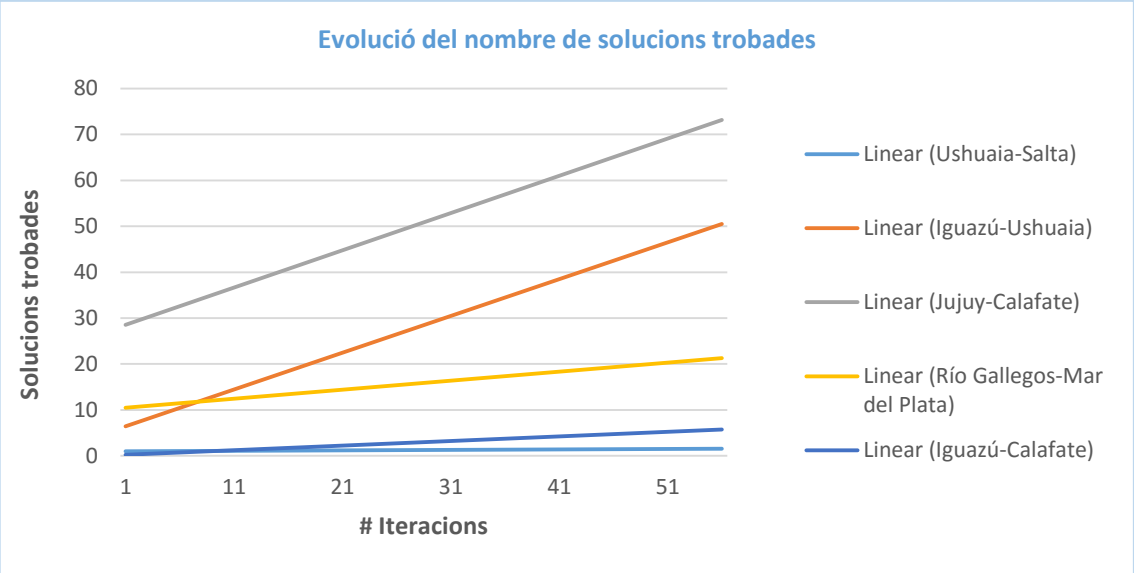

Diagrama 8. Test d'aprenentatge global. Evolució del nombre de solucions trobades.

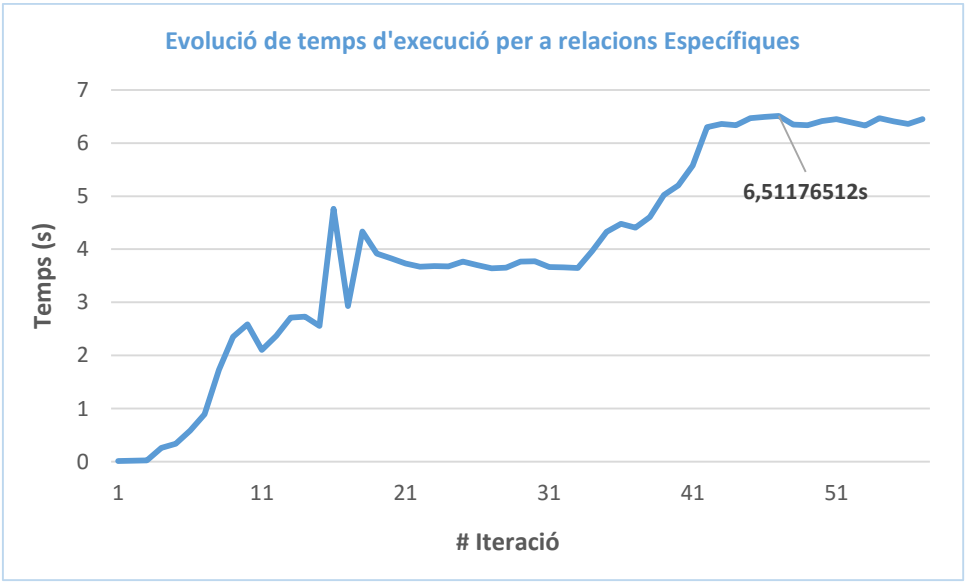

Diagrama 9. Test d'aprenentatge global. Evolució del temps mitjà d'execució de les 5 parelles amb combinacions de relacions Específiques.

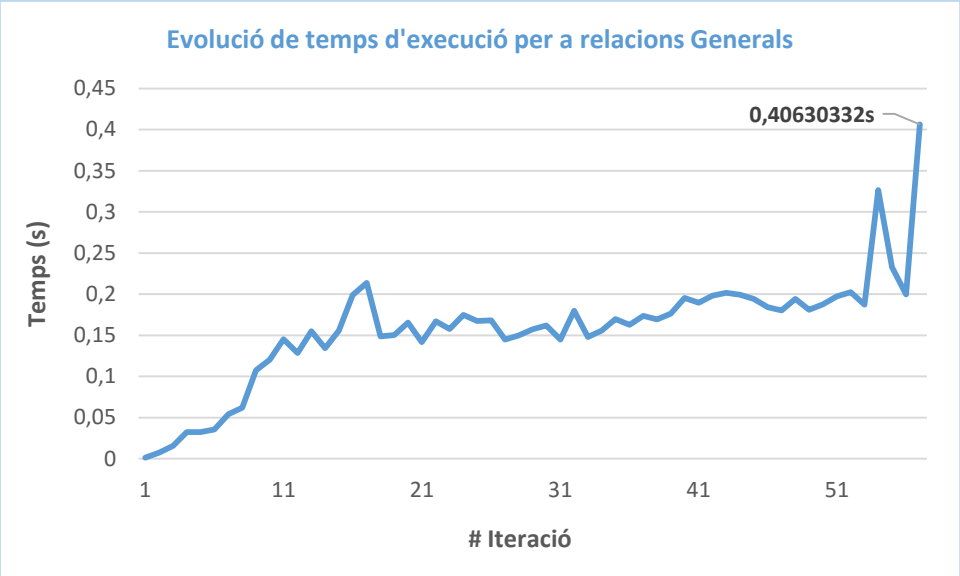

Diagrama 10. Test d'aprenentatge global. Evolució del temps mitjà d'execució de les 5 parelles amb combinacions de relacions Generals.

JUSTIFICACIÓ DE CONSTANTS

Com es va explicar a l'apartat d'Implementació, les constants per a *k* i per al factor de ramificació d'A* per a relacions Específiques, són 15 i 100 respectivament, i són els que s'han utilitzat al test anterior i les que el sistema està utilitzant actualment. Aprofitant l'estat de la base de dades després d'aquest últim test, tornarem a realitzar unes iteracions sobre el mateix ignorant l'etapa d'aprenentatge, i utilitzant valors diferents per aquestes dues constants.

Primer fixarem *k* a 15 i realitzarem proves per a diferents valors pel factor de ramificació. Per cada parella de valors, comprovarem les mitjanes dels temps de resposta totals i dels temps d'execució de K* per a relacions Específiques. A la Taula 13 es mostren els resultats de les execucions amb un valor de *k* fixat a 15, i variant el factor de ramificació. Podem veure clarament com, a l'utilitzar factors de ramificació superiors a 100, els temps de K*, i en general els temps de resposta, són massa alts per poder plantejar-nos utilitzar factors de ramificació superiors, encara que ens donessin resultats més econòmics (tot i que no és el cas).

En quant al valor de *k*, no cal augmentar-ho massa per adonar-nos de que és inviable utilitzar valors superiors a 15. Si l'augmentem només, per exemple, a 25, els temps de resposta poden superar el minut, simplement perquè la xarxa es satura al intentar realitzar tantes peticions a servidors externs de manera paral·lela. Evidentment això és degut principalment a la qualitat

limitada de la connexió del nostre servidor, i potser amb una millor connexió es podria estudiar utilitzar valors més grans per a *k*; tot i així, això no sembla gaire necessari, ja que trobar més combinacions (pitjors) al graf no serviria més que per oferir una major quantitat d'ofertes a l'usuari que molt probablement no millorarien ni en temps, ni en transbords, ni en preu a les que es poden trobar amb les 15 més econòmiques, que creiem que són suficients.

K	***Factor de ramificació***	**Temps mitjà de resposta**	**Temps mitjà de K* (rels específiques)**
15	100	20s	6.7s
15	200	26s	14.64s
15	500	33s	25.7s
15	1000	34.2s	26s

Taula 13. Justificació de constants. Resultats al variar el factor de ramificació.

7.2.2 TEST SOBRE EL *FRONT-END*

Seguidament es realitzarà una prova sobre la interfície web per verificar el seu funcionament. A més a més, els resultats obtinguts es compararan amb els de la *Travel Search Engine* més popular a l'Argentina, *Despegar.com* (que només cerca viatges amb avió).

Amb el mateix estat de la base de dades deixat pel test anterior, començarem repetint una de les cerques d'aquest i demanarem a la interfície que busqui els viatges entre la parella *Jujuy→Calafate* pel dia 10 de juny, però en aquest cas per a dos adults i un nen d'entre 5 i 11 anys. La pantalla d'introducció de dades amb els camps omplerts es mostra a la Figura 46. Als resultats obtinguts els hi apliquem un filtre per a que només es mostrin viatges amb un preu màxim de 10.000$ i que tinguin com a molt dues escales. Els resultats després dels filtres es mostren a la Figura 47; noti's com els preus més econòmics de la cerca (preus per adult) coincideixen amb els obtinguts al test d'aprenentatge (Diagrama 5). Si comparem aquests resultats amb els disponibles a través de *Despegar.com*, veiem com els nostres són molt més econòmics (el viatge més econòmic per a la mateixa cerca a la seva web és de 15.518$). A

continuació canviarem la cerca a partir del botó desplegable de l'esquerra per buscar els viatges de la parella *Río Galledos→Mar del* Plata, també pel dia 10 i per dos adults i un nen. Un cop obtinguts els resultats, els ordenem per hora de sortida i els filtrem per a que només es mostrin els que surtin abans de les 11 del matí i arribin en un dia o menys (Figura 49). Veiem com els preus per adult més econòmics coincideixen una altra vegada amb els del test d'aprenentatge (Diagrama 6), i com tornem a superar als preus de *Despegar.com*, on el viatge més econòmic costa 11.372$. Si, per exemple, decidim comprar el primer viatge mostrat, podem clicar a "Ver" per veure el desglossament de preus i tenir accés als *links* a les pàgines externes per obtenir els respectius bitllets (Figura 50).

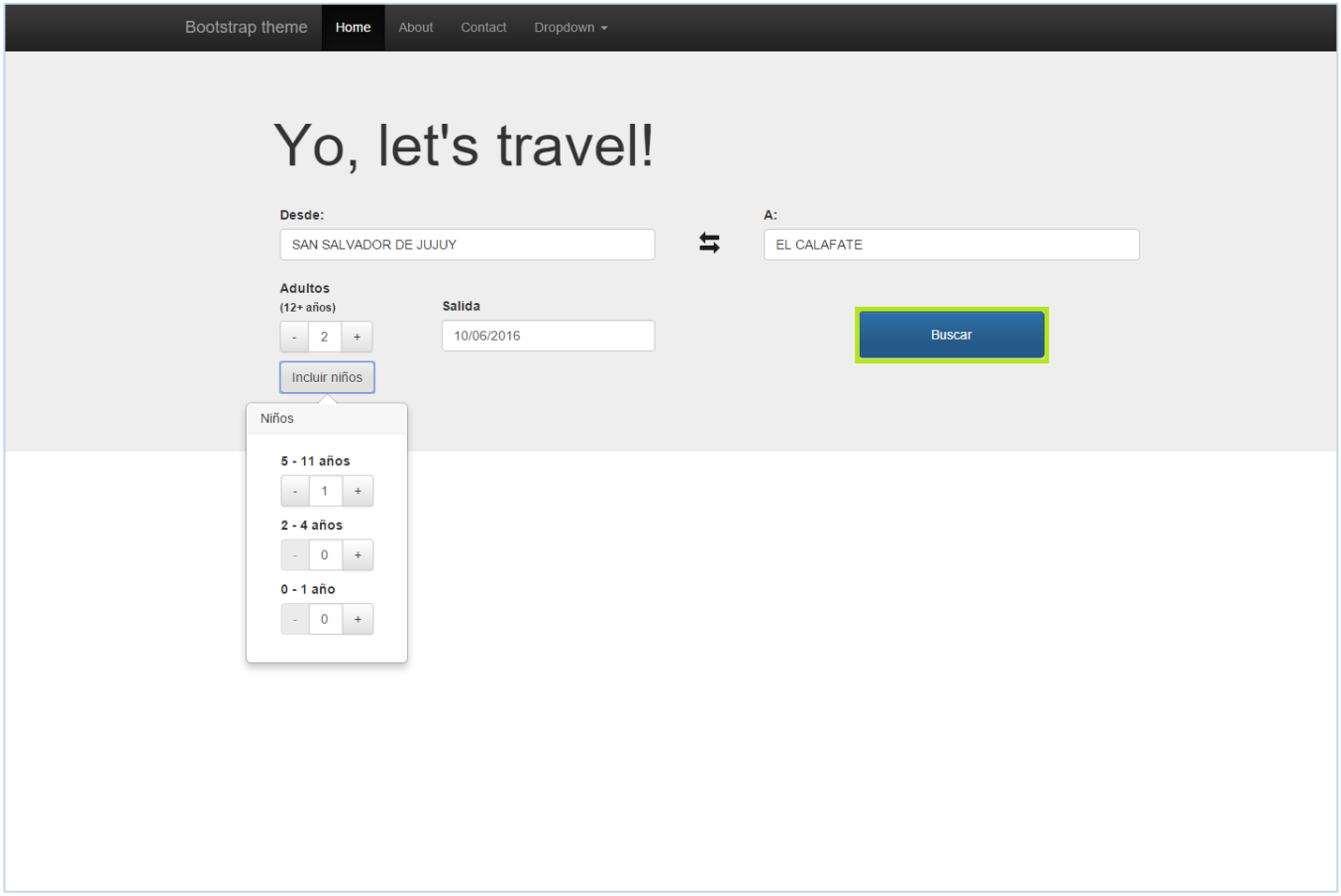

Figura 46. Test sobre la interfície. Pantalla d'introducció de dades.

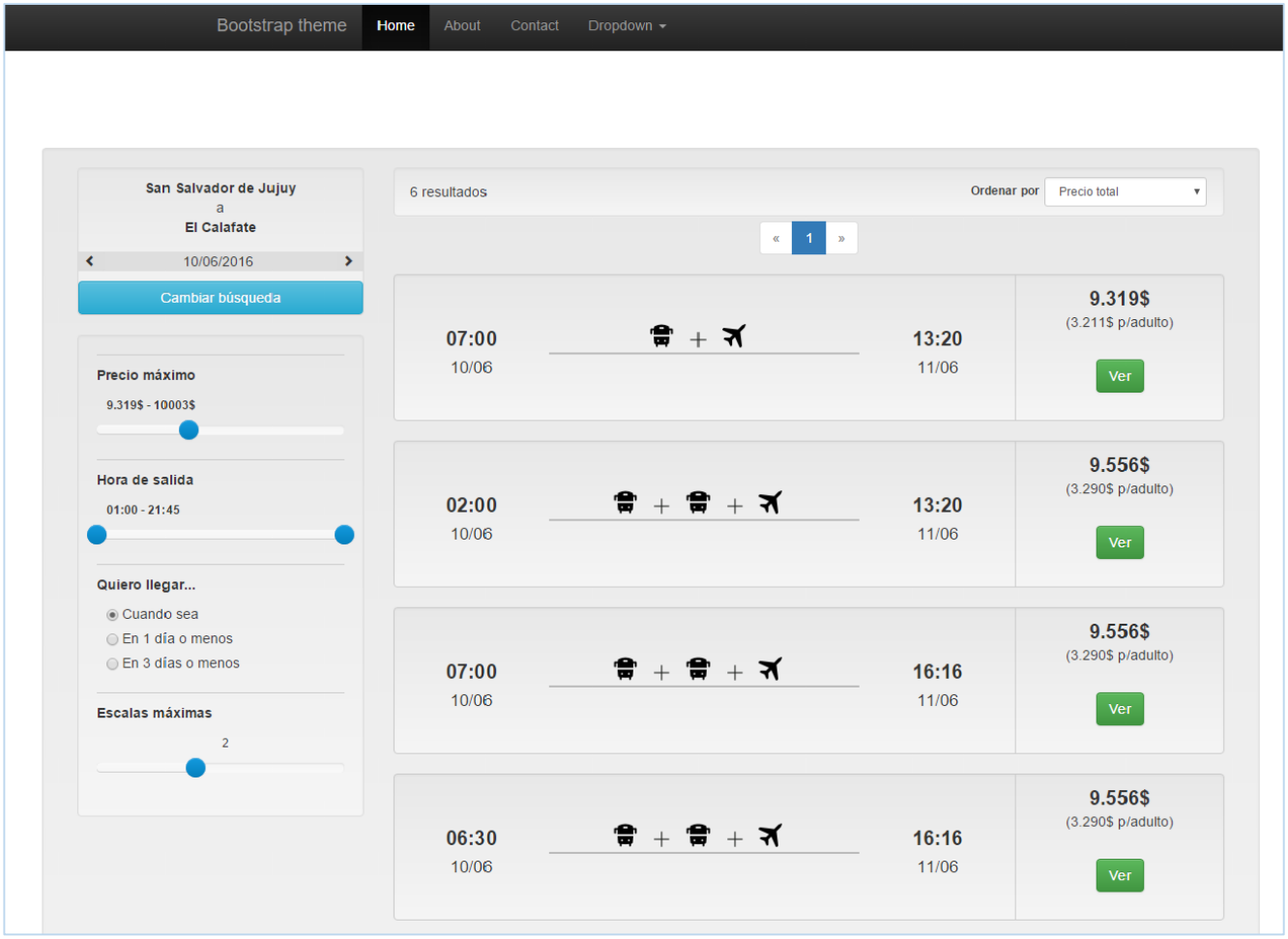

Figura 47. Test sobre la interfície. Pantalla de mostra dels resultats.

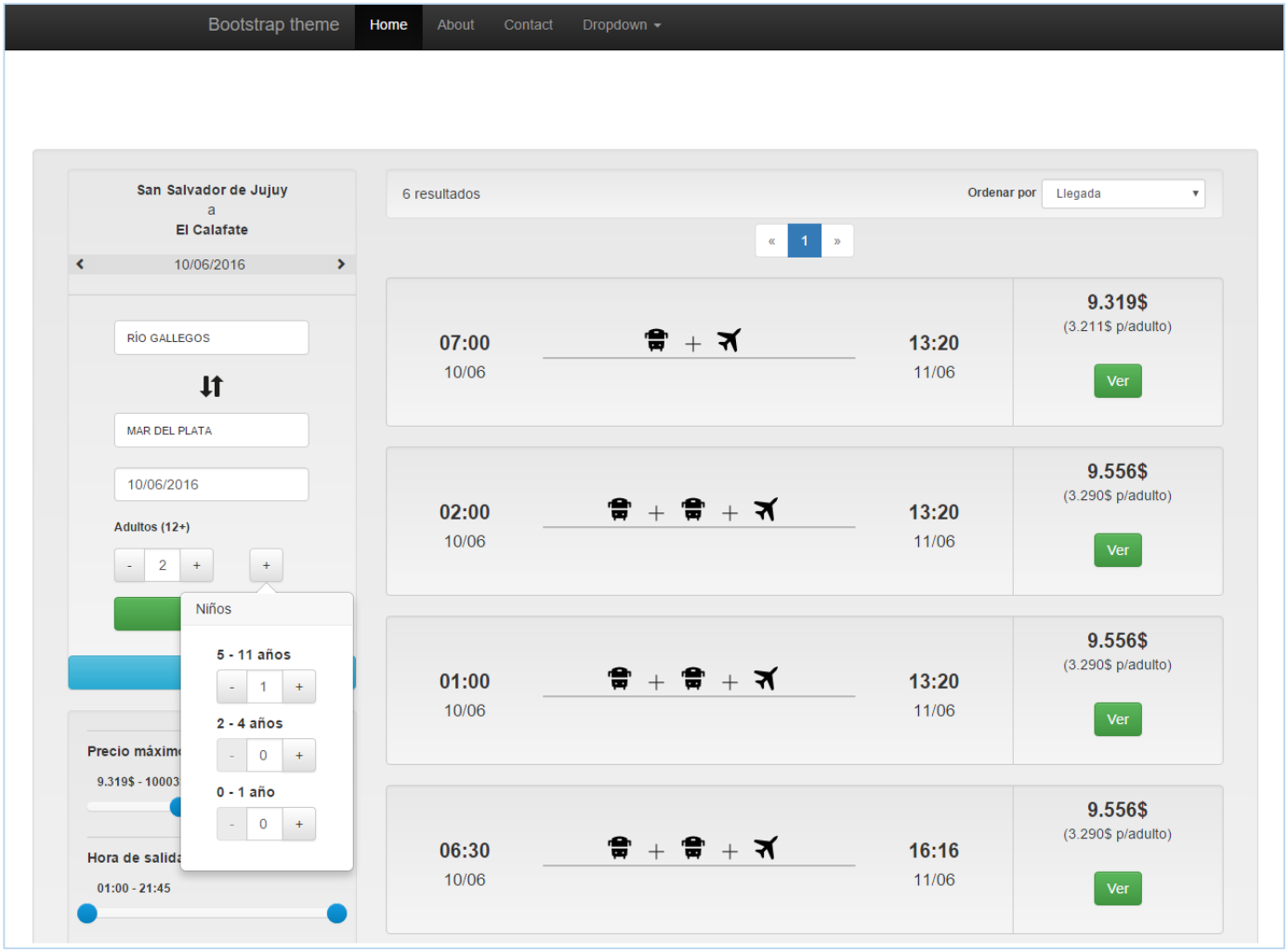

Figura 48. Test sobre la interfície. Pantalla de canvi de cerca.

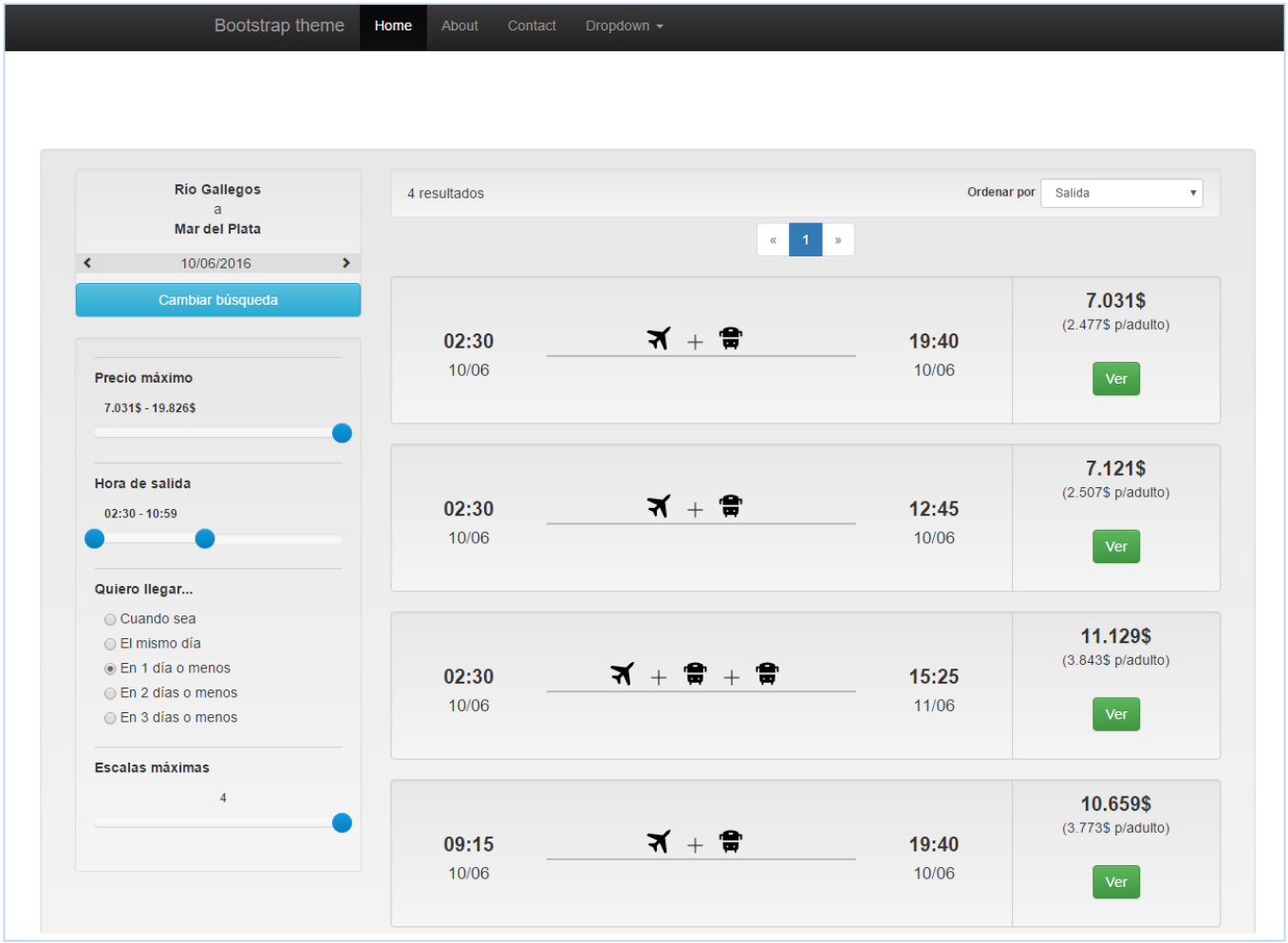

Figura 49. Test sobre la interfície. Pantalla de mostra dels resultats (segona cerca).

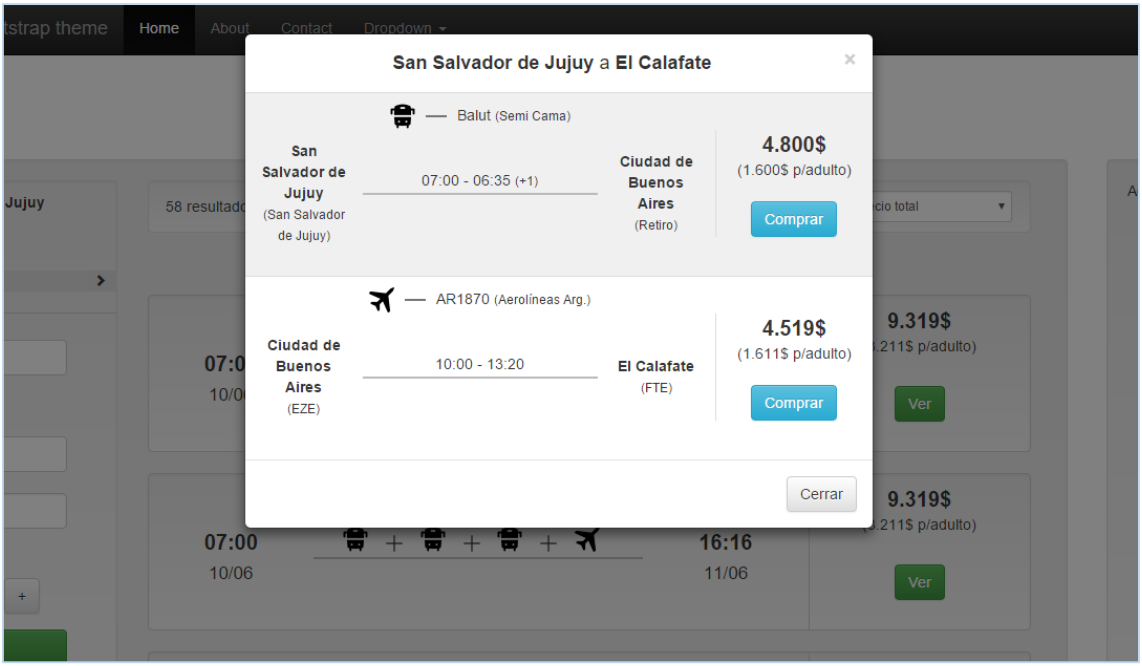

Figura 50. Test sobre la interfície. Pantalla de desglossament de preus.

7.2.3 LA QÜESTIÓ TEMPORAL

Com es va poder veure al test global d'aprenentatge, els temps de resposta per obtenir els viatges amb combinacions es troben, en general i amb una base de dades força densa, entre els 20 i els 30 segons. Aquests, encara que es troben dins del rang dels temps de resposta que s'esperaven de l'aplicació a la planificació del projecte, sens dubte no són gaire adequats per a una pàgina web. Tenint en compte que els principals responsables d'aquesta despesa temporal són, conjuntament, el nostre servidor i els servidors externs, principalment el de *Aerolíneas Argentinas*, la única solució que podem realitzar a curt termini és trobar un servidor extern més ràpid que aquest per cercar els viatges amb avió, ja que la possibilitat d'aconseguir un servidor amb una millor connexió és remota ara mateix. Concretament, seria molt interessant poder reduir aquests temps d'execució a com a màxim 15 segons de resposta; és a dir, en general, com a molt 5 segons per obtenir les combinacions reals després de la cerca de K*.

8. ANÀLISI ECONÒMICA I SOSTENIBILITAT

8.1 ANÀLISI ECONÒMICA

En d'aquest apartat tractarem de realitzar una estimació econòmica del conjunt del projecte, tenint en compte tots els recursos detallats als apartats anteriors. Dividirem aquesta estimació en quatre apartats: costos directes per activitat, costos indirectes, contingències i imprevistos. Les amortitzacions es tindran en compte dins de les partides a les que siguin necessàries, i no representaran una partida apart.

Tots els elements que es tindran en compte per a realitzar les partides deriven de les tasques descrites al diagrama de Gantt de l'apartat de Planificació Temporal.

8.1.1 COSTOS DIRECTES PER ACTIVITAT

PRESSUPOST DE RECURSOS HUMANS

Encara que a la realitat el cost dels treballadors (dissenyador, desenvolupador i beta tester) és nul, ja que a la pràctica no hi ha cap contracte laboral, es farà una simulació dels contractes, aplicant els sous estàndards per a cadascun d'ells (Taula 14).

Treballador	Hores	Preu/hora	Preu total
Dissenyador	155	35 €	5.425 €
Desenvolupador	220	35 €	7.700 €
Beta tester	140	25 €	3.500 €
TOTAL	515		**16.625 €**

Taula 14. Partida de recursos humans.

PRESSUPOST DE HARDWARE

El *hardware* serà una de les principals despeses al projecte, principalment perquè és l'eina de treball essencial a totes les etapes d'aquest. Cadascun dels dispositius *hardware* utilitzats al projecte es van detallar a l'apartat de Recursos. A continuació especificarem (Taula 15), per cada dispositiu, quin és el seu cost dins del projecte, tenint en compte el seu temps d'amortització i el seu cost total de mercat (sense tenir en compte l'IVA, que s'aplicarà més tard).

Producte	Preu	Unitats	Vida útil	Amortització
PC	701,65 €	1	3 anys	77,96 €
Monitor	90,08 €	1	3 anys	10 €
Portàtil	332,09 €	1	3 anys	36,89 €
Bases de dades	0 €	2	-	0 €
TOTAL	1123,82 €			**124,85 €**

Taula 15. Partida de Hardware.

PRESSUPOST DE SOFTWARE

El pressupost del *software* serà nul (Taula 16), ja que tot el *software* que s'utilitzarà serà o lliure o de llicència gratuïta atorgada a estudiants per *Microsoft* o per altres plataformes (*Sublime Text 3*).

Producte	Preu	Unitats	Vida útil	Amortització
Windows 10/8.1	0 €	2	2 anys	0 €
Opensuse Tumbleweed	0 €	1	2 anys	0 €
Sublime Text 3	0 €	1	2 anys	0 €
Word 2013	0 €	1	2 anys	0 €
TOTAL	0 €			**0 €**

Taula 16. Partida de Software.

8.1.2 COSTOS INDIRECTES

DESPESES GENERALS

A les despeses generals del projecte hi tindrem en compte tres conceptes: la despesa en electricitat per utilitzar el *hardware* i altres elements, com poden ser les llums; la despesa del telèfon pels serveis d'internet; i el cost del transport per anar a veure al director del projecte a les reunions (Taula 17).

Producte	Preu	Unitats	Preu total
Electricitat	0,11 €/Kwh	480 (aprox.)	52,80 €
Telèfon	40 €/mes	4	160 €
Transport	20 €/mes	4	80 €
TOTAL			**292, 80 €**

Taula 17. Partida de Despeses Generals

8.1.3 CONTINGÈNCIES

A més del cost fix del projecte, existeixen altres situacions conegudes que poden posar en risc l'estabilitat pressupostària del mateix. Aquestes situacions, sovint incontrolables, s'han de pressupostar per a evitar que se'ns dispari el pressupost.

Al nostre projecte tenim principalment tres possibles contingències:

- **Base de dades pel sistema**. Encara que és molt improbable que succeeixi dintre del període de 4 mesos de desenvolupament del projecte, podem arribar a necessitar utilitzar un pla de pagament per a la nostra base de dades en forma de graf si decidim mantenir-la a un servidor extern. Els plans gratuïts solen ser suficients al començament dels projectes, però per uns 10€ més (estimats, ja que encara no s'ha dissenyat la base de dades i no s'han buscat proveïdors) podem augmentar la capacitat del servidor.
- **Base de dades per a la pàgina *web***. De manera similar, per la base de dades de la pàgina *web* es pot arribar a necessitar un pla de pagament, a un preu semblant.
- **Augment del temps de treball.** Com hem dit a l'apartat de Planificació Temporal, el temps d'implementació es pot arribar a disparar en, aproximadament, 20 hores. Això comportarà unes despeses extres d'electricitat.

L'import del pla de contingència es detalla a continuació (Taula 18).

Contingència	Preu	Unitats	Preu total
Base de dades CBR	10 €	1	10 €
Base de dades *web*	10 €	1	10 €
Electricitat	0,11 €/Kwh	20	2,20 €
TOTAL			**22,20 €**

Taula 18. Partida de contingències.

Com que les contingències referents a les bases de dades poden sorgir encara després de la realització del projecte per la possible fama de la plataforma, si aquestes no arribaren a sorgir en el període de quatre mesos, els diners del pressupost es destinarien a finançar-les.

8.1.4 PRESSUPOST TOTAL

A continuació es presenta el sumatori de les diferents partides descrites anteriorment (Taula 19).

Partida	**Preu**
Recursos humans	16.625 €
Hardware	124,85 €
Software	0 €
Despeses generals	292, 80 €
Contingències	22,20 €
TOTAL	**17.064,85 €**

Taula 19. Pressupost total.

Tot i això, al projecte poden sorgir imprevistos impossibles de calcular ara mateix, ja que són fruit del mateix desenvolupament del projecte. Aquests imprevistos poden arribar a suposar una gran càrrega pressupostària si no els tenim en compte abans de començar amb el treball. Per a controlar econòmicament aquests possibles esdeveniments, es destinarà una suma extra equivalent al 10% de l'import calculat del projecte (Taula 20).

Import	**Percentatge**	**Import total**
17.064,85 €	10 %	**18.771,34 €**

Taula 20. Pressupost total amb imprevistos.

A continuació (Taula 21), s'aplica l'impost corresponent de l'IVA al total de l'import del projecte. El resultat serà el pressupost final calculat pel projecte.

Import	Impost	Import total
18.771,34 €	21 %	**22.713,32 €**

Taula 21. Pressupost total amb impostos.

8.2 ANÀLISI DE SOSTENIBILITAT

Seguidament, a la Taula 22, es presenta la matriu de sostenibilitat del projecte amb les puntuacions corresponents.

	PPP	Vida útil	Riscos
Ambiental	9	17	-1
Econòmic	8	17	-1
Social	9	18	0
Rang sostenibilitat	26	54	-2
TOTAL	**78**		

Taula 22. Matriu de sostenibilitat

8.2.1 DIMENSIÓ ECONÒMICA

Els costos establerts anteriorment per al projecte són justificats, ja que cap partida està fora de les dimensions que suposa la realització d'una pàgina *web*, de manera general. Tot i que, evidentment, s'ha de tenir en compte que el cost econòmic del projecte no acaba aquí, sinó que aquest requerirà actualitzacions que necessitaran de més mà d'obra i recursos, a més de l'actualització periòdica de les bases de dades, tant per qüestió d'obsolescència com per qüestió d'augment de tràfic al servidor.

Tenint en compte que la majoria de projectes *web* d'aquestes dimensions solen tenir un pressupost superior als 50.000 €, un pressupost de menys de la meitat ho fa completament competitiu amb la resta del mercat, i realment seria complicat realitzar-ho encara amb menys recursos. A més, en general s'ha repartit bé el temps de les tasques dins del projecte segons la seva importància: tant el temps per a la part de disseny com per a la part de *testing* són proporcionals al temps de desenvolupament.

8.2.2 DIMENSIÓ AMBIENTAL

En l'àmbit ambiental, el projecte consumirà el que qualsevol projecte amb el propòsit de crear una plana *web* consumiria, o inclús menys, en concepte d'electricitat, principalment. Aquesta electricitat s'ha de produir a les centrals d'energia, molt probablement no de manera 100% neta, així que, ens agradi o no, sempre estarem deixant una petjada ecològica. De tota manera, no s'està fent un ús exagerat o irresponsable de l'energia, i possiblement s'utilitzaria pràcticament la mateixa si el projecte no formés part d'un TFG.

Finalment, creiem fermament que la nostra plataforma servirà per a fomentar el viatge en transport col·lectiu entre els turistes (avió i bus), i fer que aquests utilitzin menys el transport privat al veure que els preus poden ser més econòmics amb altres opcions. Així, de manera indirecta, podríem ajudar a reduir la petjada ecològica de la població sobre el món.

8.2.3 DIMENSIÓ SOCIAL

En quant a l'aportació social del projecte, com ja es va comentar anteriorment quan s'introduí, l'aplicació pot esdevenir molt positiva pel turisme a l'Argentina, que està en una situació de creixement tant per part del turisme estranger com del turisme local. Una plataforma com la que es proposa facilitaria el moviment dels turistes dins del país, fent més còmode i a la vegada més atractiu visitar les meravelloses zones que s'hi troben.

A més, com també es va comentar, no existeix una plataforma similar exclusiva per l'Argentina, i menys on es combini tant el transport amb avió com amb bus. A un país tan gran amb tantes combinacions de transport, és molt necessari tenir una eina que et proporcioni informació sobre la manera més econòmica o ràpida de viatjar d'una ciutat a l'altra.

Per últim, assumint que una plataforma així farà que molta més gent es plantegi viatjar, és important notar que aquesta tindrà una gran aportació humana, ja que viatjant les persones interpreten noves maneres de fer i es fomenta de manera indirecta el concepte de cultura global, que tant cal avui dia. Hi ha gent a l'Argentina que no sap com viu la gent a l'altra punta del seu país, amb costums totalment diferents als seus. Si s'aconsegueix un nivell de popularitat suficient, es podria ajudar molt en aquest sentit a la societat en general, i a l'Argentina en concret.

9. PLANIFICACIÓ TEMPORAL

En aquesta secció exposarem la planificació temporal del projecte, incloent-hi la divisió d'aquest en tasques específiques, i l'ajustament i viabilitat d'aquestes dins el temps estipulat per realitzar el projecte, fent una aproximació coherent del temps necessari per acabar cadascuna. Per cada tasca, es descriurà el seu abast dins del projecte i què s'hauria d'haver assolit quan es finalitzi. Es proporcionarà un diagrama temporal de Gantt on es veurà reflectida la durada de cadascuna de les tasques (Figura 51), a més de la seqüència lògica necessària degut a les dependències entre elles. S'especifiquen, a més, els recursos utilitzats durant el transcurs del projecte i quines tasques els utilitzen. Finalment es comenten les desviacions que hi ha hagut a la planificació en el transcurs del projecte.

El projecte va començar el dia 22 de febrer i hauria de concloure el dia 20 de juny (16 setmanes). Dins d'aquesta finestra temporal s'haurien de dur a terme les tasques que en aquest apartat es descriuen.

9.1 DESCRIPCIÓ DE LES TASQUES

A continuació s'especifiquen les diferents tasques dins del projecte, en ordre temporal d'elaboració.

9.1.1 PLANIFICACIÓ DEL PROJECTE

Aquesta tasca forma part de la fita inicial. És la primera tasca que cal realitzar, ja que és on s'especifica quin és l'abast del projecte, el seu objectiu i com s'arribarà a aquest. Es divideix en les següents etapes:

- Abast i contextualització.
- Planificació temporal.
- Gestió econòmica i sostenibilitat.

9.1.2 ANÀLISI I DISSENY DEL PROJECTE

El propòsit d'aquesta tasca no és altre que trobar el disseny més raonable pel projecte, tenint en compte l'objectiu o el producte final que es vol aconseguir, les capacitats del desenvolupador i el temps per a realitzar-ho. S'hauran d'analitzar les diferents alternatives disponibles per als diferents mòduls dins del projecte, utilitzant la literatura disponible i el coneixement propi. En resum, s'haurà de passar d'una idea de projecte a una especificació clara i robusta d'aquest, per a posteriorment tenir clar com s'ha d'implementar cadascun d'aquests mòduls.

La tasca es dividirà en les següents etapes:

- *Elaboració dels casos d'ús.* Especificar com interaccionarà l'usuari amb el sistema i quina seqüència d'accions farà el sistema per a cada *query* de l'usuari.
- *Anàlisi i disseny de CBR i els algorismes de cerca*. Estudi de l'aplicació del cicle CBR a la nostra situació, de la millor base de dades pel sistema i de quins algorismes ens poden ser útils per realitzar la cerca sobre el graf.
- *Llenguatge de programació i obtenció de dades.* Estudi del millor llenguatge de programació per a la implementació del nostre *back-end*, així com l'especificació del mètode per realitzar l'*scraping* a les diferents pàgines web d'on haguem d'extreure la informació relativa als viatges.
- *Unificació de CBR i obtenció de dades*. Dissenyar l'estructura final del *back-end*; és a dir, com es comunicarà el cicle CBR amb l'*scraping* per a assolir una resposta final per l'usuari.

Amb aquestes quatre etapes completades, ja tindrem tot enllestit per començar a implementar el nostre projecte.

Cal anotar, però, que el disseny de la pàgina web (*front-end*) es deixa a fora d'aquest disseny inicial del projecte, ja que, tot i que forma part d'aquest, preferim dissenyar la pàgina just abans d'implementar-la, degut a que poden haver-hi endarreriments que facin que un disseny que es pensi a l'inici no sigui viable més endavant per manca de temps, i no és important per a

l'elaboració del *back-end*. En altres paraules, i com ja s'ha dit a l'abast del projecte, la complexitat al disseny de la pàgina web dependrà de si el projecte es desenvolupa o no a la velocitat esperada.

També és adient puntualitzar que aquesta segona tasca es pot començar un cop acabada la planificació temporal, ja que no influirà massa el coneixement sobre l'impacte ambiental o econòmic sobre les decisions a l'hora de dissenyar del sistema.

9.1.3 IMPLEMENTACIÓ

Aquesta és la tasca que hauria d'ocupar la majoria del temps del projecte. És on s'implementarà tot el dissenyat a la tasca anterior, seguint el mateix ordre dels mòduls, a més d'afegir les corresponents etapes de *testing*.

Així doncs, aquesta tasca es dividirà en les següents etapes:

- *Implementació de la base de dades, CBR i els algorismes de cerca.* Aquesta etapa ocuparà la gran part del temps. S'haurà de construir la base de dades en forma de graf i posteriorment implementar sobre ella el cicle de CBR que utilitzarà els corresponents algorismes de cerca per trobar els camins dins la base de dades.
- *Testing*. Després d'haver acabat la codificació principal del sistema, haurem d'idear uns jocs de proves per tal de que estiguem convençuts de que funciona correctament. Aquests jocs de proves es basaran en simulacions de *queries* de l'usuari i han de resultar en casos concrets trobats per resoldre-les.
- *Implementació d'Scrapers*. Els *scrapers* seran una part fonamental del sistema i poden arribar a donar molta feina, depenent de la complexitat de les pàgines web. En un principi s'hauran de fer tres *scrapers* capaços d'obtenir informació, respectivament, de les pàgines *www.aerolineasargentinas.com* (avions), *www.lan.com* (avions) i *www.plataforma10.com* (busos). El *testing* en aquesta etapa no serà tant important com a l'anterior, ja que no hi ha cap complexitat algorísmica darrera que ens pugui fer dubtar del seu correcte funcionament.

- *Unificació de CBR i Scrapers*. Un cop tinguem tant el sistema principal com els *scrapers* funcionant correctament, serà el moment d'implementar el comunicador entre aquests dos mòduls per a que els camins trobats a la base de dades puguin ser contrastats amb informació en temps real. Aquesta etapa no hauria de comportar gaire temps, ja que realment no cal fer res nou. Un cop haguem fet la unificació, ja tindrem acabat el *back-end* del nostre projecte.
- *Testing del back-end*. Per assegurar-nos de que el *back-end* funciona correctament, haurem de fer proves similars a les fetes al *testing* anterior, simulant *queries* de l'usuari, però que sens dubte ens prendran més temps ja que estem abastint un sistema molt més ampli.
- *Disseny i implementació web*. Un cop el *back-end* funcioni com esperàvem, serà hora de posar-se a, primer dissenyar, i després implementar la pàgina web. Aquest disseny serà simple o complex, com s'ha dit abans, depenent de si el projecte s'ha pogut portar a terme amb la celeritat esperada. El procés d'implementació també suposarà la connexió de la pàgina web amb el *back-end* implementat anteriorment.
- *Tests finals*. Acabada la pàgina web, començarà una etapa intensiva de *testing* per a corroborar que tot funciona de manera correcta, i que efectivament els usuaris reben bones recomanacions de combinacions de viatges per a les seves *queries* a partir de la interfície web.

9.1.4 TASCA FINAL

Aquest serà el moment de comprovar que tot funciona correctament, potser provant a diferents sistemes operatius, navegadors, etc. A més, s'haurà d'enllestir la memòria del projecte i preparar la entrega i defensa final.

9.2 DURACIÓ DE LES TASQUES

A la Taula 23 s'indiquen els temps estimats de cadascuna de les tasques especificades a l'apartat anterior.

Tasca	Temps de realització (hores)
Planificació del projecte	40
Anàlisi i disseny del projecte	40
Implementació de la base de dades, CBR i els algorismes de cerca	75
Testing	25
Implementació d'*Scrapers*	60
Unificació de CBR i *Scrapers*	10
Testing del *back-end*	35
Disseny i implementació web	150
Tests finals	30
Tasca final	50
TOTAL	**515**

Taula 23. Duració de les tasques

	Nombre	Duración	Inicio	Fin	Predecesoras	Recursos
1	Abast i contextualització	4d	22/02/2016	29/02/2016		
2	Planificació temporal	2.04d	01/03/2016	05/03/2016	1	
3	Gestió econòmica i sostenibilitat	2d	06/03/2016	10/03/2016	2	
4	Elaboració dels casos d'ús	2d	06/03/2016	10/03/2016	2	
5	Anàlisi i disseny de CBR i els algorismes de cerca	2.67d	11/03/2016	15/03/2016	4	
6	Llenguatge de programació i obtenció de dades	1.33d	15/03/2016	18/03/2016	5	
7	Unificació de CBR i obtenció de dades	1.38d	19/03/2016	20/03/2016	6	
8	Implementació de la base de dades, CBR i els algorismes de cer	7.04d	21/03/2016	03/04/2016	7	
9	Testing	1.33d	04/04/2016	07/04/2016	8	
10	Implementació d'Scrapers	7d	08/04/2016	20/04/2016	9	
11	Unificació de CBR i Scrapers	2.25d	21/04/2016	24/04/2016	10	
12	Testing del back-end	1.67d	24/04/2016	29/04/2016	11	
13	Disseny i implementació web	19.71d	30/04/2016	05/06/2016	12	
14	Tests finals	1.67d	05/06/2016	10/06/2016	13	
15	Tasca final	4.71d	11/06/2016	19/06/2016	14	

Figura 51. Diagrama de Gantt. Planificació.

9.3 VALORACIÓ D'ALTERNATIVES

Tot i que en general no esperem que hi hagi imprevistos, sempre pot ocórrer que una tasca qualsevol ens acabi donant més feina, i per tant, més hores de treball. Aquests imprevistos o desviacions, si ocorren, ho faran principalment en tasques d'implementació. Tot i així, com ja es va comentar a l'apartat de metodologia, s'intentarà evitar al màxim aquests errors per mitjà de la creació constant de prototips que seran posats a prova de manera seguida.

En general, es considera que les petites desviacions en la càrrega de treball com a molt podrien fer sumar 10 hores a les tasques d'implementació (implementació de CBR i implementació d'*scrapers*), fent un temps total del projecte de fins a 535 hores, a repartir entre 16 setmanes: 33 hores per setmana; un temps totalment assolible per a poder finalitzar el projecte. Tot i així, si, en un utòpic, aquesta acumulació de desviacions fos tan gran que veiéssim que ens acabés faltant temps per completar totes les tasques, com també s'ha repetit anteriorment, es traurà temps de la tasca de disseny i implementació de la web, ja que és la que menys importància té pel correcte funcionament del sistema.

9.4 RECURSOS

A continuació s'esmenten els recursos necessaris per a dur a terme el projecte, juntament amb les tasques que els utilitzaran. Els separem en recursos *hardware* i recursos *software*.

9.4.1 RECURSOS *HARDWARE*

- PC, amb les següents prestacions: *Intel core* i5-4670, 8GB RAM, 2TB de disc magnètic + 120GB de disc sòlid, *nVidia GeForce* GTX 750): utilitzat a totes les tasques.
- Monitor *Asus VS229HA 22'' LED*: utilitzat a totes les tasques.

- Ordinador portàtil, amb les següents prestacions: *Interl core* i3-4030U, 4GB RAM, 400GB de disc magnètic, *nVidia GeForce* 820M): utilitzat a totes les tasques.
- Bases de dades remotes: utilitzat a implementació de CBR i a disseny i implementació web.

9.4.2 RECURSOS *SOFTWARE*

- *Windows 10/8.1*: utilitzat a totes les tasques.
- *Opensuse Tumbleweed*: utilitzat a totes les tasques.
- *Sublime text 3*: utilitzat a les tasques d'implementació (implementació de CBR, implementació d'*scrapers*, unificació de CBR i *scrapers* i disseny i implementació web).
- *Word 2013*: utilitzat a totes les tasques.

9.5 DESVIACIONS SOBRE LA PLANIFICACIÓ

Durant el transcurs del projecte, hi ha hagut certes variacions a la planificació inicial, principalment a l'etapa d'Implementació. Tot i que no s'ha hagut d'afegir cap tasca inesperada, algunes d'elles han resultat ser més ràpides de realitzar, i per contra d'altres han resultat ser més lentes. A més a més, l'ordre d'algunes va acabar canviant, i d'altres es van acabar realitzant en diferents períodes de temps. A continuació es comenten els diferents canvis sobre la planificació inicial, que es poden veure reflectits al digrama de Gantt de la Figura 52.

- **La Implementació dels *scrapers* es realitzà abans de CBR i els algorismes de cerca**. Aquesta decisió va ser justificada, principalment, per la dificultat de les tasques i el desconeixement del llenguatge de programació. La implementació dels *scrapers* era una tasca d'implementació molt més senzilla que no pas la implementació del nucli del *back-end*. Això, si s'hagués estat programant amb un llenguatge de programació ben conegut com Java o Python, no hagués sigut un gran problema, però la utilització d'un llenguatge amb el qual no s'havia treballat abans al qual calia agafar-li el ritme, va fer pensar que potser era molt millor començar per la tasca més simple d'implementar, per estar més

o menys acostumat al llenguatge quan es comencés la part més important de la implementació. A més a més, la implementació dels *scrapers* ens va portar uns 10 dies de feina, no 7 com s'havia pensat.

- **La Implementació de CBR i els algorismes de cerca va acabar sent molt més llarga del que es pensava, mentre que el disseny i la implementació web va acabar sent molt més curt.** La implementació del nucli de l'aplicació (CBR i els algorismes de cerca) va resultar ser molt més llarga del que es va pensar en un principi; aquesta va acabar ocupant tot el mes d'abril, després d'acabar amb els *scrapers* (incloent-hi les tasques de *testing* i d'unificació). Afortunadament, la tasca de disseny i implementació web (que abastava un disseny inicial, com s'explica al següent punt) va durar un terç del que es va pensar inicialment, ocupant només els deu primers dies de maig.
- **Divisió de la tasca final i del disseny i implementació web.** Com que es veia que la implementació de la interfície seria més ràpida del que s'havia previst, es va decidir començar amb la memòria just quan es tingués un disseny funcional i bàsic d'aquesta, per assegurar-nos que tindríem temps de redactar-la i repassar-la sense presa. Un cop s'acabés la memòria (cap al 7 de juny) es reprendria la implementació de la interfície web fins al dia 20, aproximadament, on s'hauria de començar la preparació de la defensa oral fins al dia 27. Així doncs, es va allargar el temps del projecte set dies més. La despesa extra d'electricitat que suposen aquests set dies no supera el pressupost d'imprevistos (veure apartat 8.1.4), així doncs no va caldre augmentar el pressupost del projecte.
- **Paral·lelització del disseny de CBR i els algorismes i la cerca del llenguatge de programació i els mètodes d'obtenció de dades.** Aquestes dues tasques, en teoria estimades a la planificació per durar, ambdues, 3 dies, es van acabar simplement realitzant de manera paral·lela, en el transcurs d'una setmana, tot i que la primera va portar força més feina que la segona.

	Nombre	Duración	Inicio	Fin	Predecesoras	Recursos
1	Abast i contextualització	4d	22/02/2016	29/02/2016		
2	Planificació temporal	2.04d	01/03/2016	05/03/2016	1	
3	Gestió econòmica i sostenibilitat	2d	06/03/2016	10/03/2016	2	
4	Elaboració dels casos d'ús	2d	06/03/2016	10/03/2016	2	
5	Anàlisi i disseny de CBR i els algorismes de cerca	3.83d	11/03/2016	18/03/2016	4	
6	Llenguatge de programació i obtenció de dades	3.75d	11/03/2016	18/03/2016	4	
7	Unificació de CBR i obtenció de dades	1.38d	19/03/2016	20/03/2016	6	
8	Implementació d'Scrapers	4.5d	21/03/2016	30/03/2016	7	
9	Implementació de la base de dades, CBR i els algorismes de cer	13.38d	31/03/2016	24/04/2016	8	
10	Testing	1d	25/04/2016	27/04/2016	9	
11	Testing del back-end	1.38d	28/04/2016	30/04/2016	10	
12	Disseny i implementació web. Disseny inicial	5d	01/05/2016	10/05/2016	11	
13	Tests finals	2.71d	11/05/2016	15/05/2016	12	
14	Tasca final. Memòria	11.67d	15/05/2016	07/06/2016	13	
15	Disseny i implementació web. Disseny final	6.5d	08/06/2016	20/06/2016	14	
16	Tasca final. Presentació	3.29d	21/06/2016	26/06/2016	15	

Figura 52. Diagrama de Gantt. Desviacions sobre la planificació.

10. CONCLUSIONS

Al llarg d'aquests quatre mesos, hi ha hagut temps suficient per desenvolupar la plataforma que es va idear al començar el projecte. Aquesta té les funcionalitats bàsiques i indispensables que es van marcar a la planificació, i, tot i que és evident que no és una versió final si no més bé un prototip, el nucli del sistema és completament funcional i es capaç d'actuar de la manera que es va pensar a l'inici, i la interfície, tot i que no està completament llesta encara, conté els elements fonamentals i és força intuïtiva i fàcil d'utilitzar. Amb tot, encara que quan s'estava planificant s'esperava tenir una plataforma totalment desenvolupada i llesta per lliurar-se a l'usuari final, podem assegurar que s'han complert bé els objectius principals establerts inicialment. Excepte l'objectiu, potser utòpic, de tenir la plataforma completament llesta i realitzar una campanya de màrqueting *online* – que es deixa per al futur -, tots els altres objectius s'han pogut complir satisfactòriament, inclús el relacionat amb els temps de resposta, mostrant els viatges directes en pocs segons, i les combinacions en, com a màxim, 20 o 30 segons. A més a més, la pàgina millora sens dubte els resultats de la competència actual, i es situa com una de les poques *Travel Search Engines* que combinen múltiples tipus de transport, i la única específica per a l'Argentina.

Tot i així, és evident que encara queda força treball per fer, principalment de cara a millorar els temps de resposta, les funcionalitats i la interfície. A continuació comentem tots els canvis que esperem introduir a la plataforma tant a curt com a mitjà o llarg termini.

10.1 TREBALL FUTUR

En el transcurs dels pròxims mesos, s'esperen afegir diverses funcionalitats extres al sistema amb l'objectiu de facilitar la cerca a l'usuari i ajudar-lo segons les característiques del seu viatge; aquestes funcionalitats comportarien tant una actualització de la implementació com de la interfície web. Algunes d'aquestes serien, per exemple, la possibilitat de triar un viatge d'anada i tornada o només d'anada, poder consultar els preus per a un viatge a un mes sencer, poder triar l'opció d'arribar a o de sortir de "qualsevol lloc", escollir múltiples orígens o múltiples destinacions... Aquestes funcionalitats no seran gaire complicades d'implementar i suposaran un gran canvi de cara a satisfer les necessitats de tots els usuaris. A més a més, s'haurien d'afegir

al graf (que actualment només compta amb les ciutats que tenen aeroport) moltes més ciutats o pobles importants per a les combinacions amb autobús, i establir un temps de transbord en relació a la distància entre les estacions/aeroports en comptes d'un de fix.

En quant a la implementació dels algorismes, una tasca que no s'ha pogut portar a terme al projecte per manca de temps, és la fixació dels valors per a la nostra funció heurística; és a dir, tant el percentatge que s'aplica sobre el cost del camí més curt trobat fins al moment (50% actualment), com el valor que es retorna en cas que no s'hagi trobat cap camí encara entre la parella de ciutats (400 actualment). Aquests dos valors s'haurien de fixar també en els pròxims mesos, mitjançant proves sobre una base de dades densament poblada, on es variïn ambdós valors i es comprovi la repercussió tant al temps d'execució com a la qualitat de les solucions trobades.

Paral·lelament, i com s'ha comentat en apartats anteriors, una cosa que no pot faltar corregir al sistema és el temps de resposta, que actualment és massa alt per poder pensar en posar la pàgina *online*. Això és degut principalment al temps de resposta dels diferents *scrapers*, en concret el d'*Aerolíneas Argentinas*, encara que també influeix la connexió pobre del servidor actual. La solució més immediata a aquest problema és directament canviar les pàgines on es fan els *scrapes* d'avions (*Aerolíneas* i *LAN*) per una *Travel Search Engine* com *eDreams* o *Skyscanner*, demanant-li'ls els viatges directes. Tot i així, també serà important a mitjà termini, i sempre i quan es tingui la plataforma llesta per sortir *online*, aconseguir un servidor més potent i amb una millor connexió a internet.

En quant a millores a llarg termini, hi ha una molt temptadora que és l'expansió de la plataforma a altres països i zones. Això serà relativament senzill en el cas de Brasil, Xile o Uruguai, on només caldria afegir els nodes de les ciutats corresponents a la base de dades i afegir algun *scraper* addicional, ja que les companyies que operen internament, com a Argentina, no són tantes. Tot i així, l'objectiu principal és poder portar la plataforma fins a Europa, i potser algun dia expandir-la per cobrir tot el món. Això ja seria molt més complicat, ja que hi entrarien en joc moltes més companyies i caldria un treball molt ampli per adaptar el mòdul de *scraping*; a més a més, el graf passaria a ser immens i caldria revisar com es comporten els algorismes implementats davant de grafs tant grans. A la vegada, també es pensen afegir altres mètodes de transport, com els creuers o la compartició de cotxe privat proporcionada per plataformes com *Blablacar.com*.

BIBLIOGRAFIA

[1]. De Marcken, C. (2003*). Computational Complexity of Air Travel Planning*. Retrieved from https://static.googleusercontent.com/media/www.itasoftware.com/es//pdf/ComplexityofArlineTravelPlanning_Carl_Sep-03.pdf

[2]. Garman, M. S. (2011). *Efficient search of supplier servers based on stored search results*. Google Patents. Retrieved from https://www.google.com/patents/US7979457

[3] Williams, G., SARRE, J., & SNIEZYK, A. (2015). *Method and server for providing a set of price estimates, such as air fare price estimates*. Google Patents. Retrieved from https://www.google.com/patents/US20150161636

[4] Saks, G. (2006). Travel: *The emergence of Meta Search*. Retrieved from https://blog.compete.com/2006/11/14/meta-search-kayak-sidestep-farechase-mobissimo-pinpoint-travel/

[5] Aamodt, A., & Plaza, E. (1994). *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*. Retrieved from http://www.idi.ntnu.no/~agnar/publications/aicom-94-old.pdf

[6] Schank, R. C. (1983). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. New York, NY, USA: Cambridge University Press.

[7] Schank, R., & Abelson, R. (1977*). Scripts, plans, goals and understanding*.

[8] Sànchez-Marrè, M. (2001). *Principles of case based reasoning*.

[9] Aljazzar, H. & Leue, S. (2011). *K*: A heuristic search algorithm for finding the k shortest paths.*

[10] Eppstein, D. (1994). *Finding the k shortest paths*.

[11] M. L. Fredman, R. E. Tarjan (1984). *Fibonacci heaps and their uses in improved network optimization algorithms.*

[12] Cade Metz. *Google programming Frankenstein is a Go*. *The Register*, maig del 2010.

ANNEXOS

COMPETÈNCIES TÈCNIQUES

- **CCO2.1: Demostrar coneixement dels fonaments, dels paradigmes i de les tècniques pròpies dels sistemes intel·ligents, i analitzar, dissenyar i construir sistemes, serveis i aplicacions informàtiques que utilitzin aquestes tècniques en qualsevol àmbit d'aplicació.** [En profunditat].

  S'ha triat aquesta competència tècnica degut a que és bàsicament el que es desenvolupa al projecte: una aplicació informàtica amb un sistema basat totalment en un paradigma d'intel·ligència artificial com Cased Based Reasoning dins de l'àmbit dels cercadors de viatges. El paradigma en qüestió, s'adapta al problema plantejat per dissenyar el nostre sistema intel·ligent, aplicant tant el coneixement obtingut a les assignatures de Computació com el que es pugui obtenir a partir de publicacions sobre la matèria.

  Evidentment, el nivell d'assoliment és "en profunditat", i s'aconsegueix mitjançant: primer, un profund anàlisi de les possibles opcions per aplicar CBR i escollint la més adient; segon, el disseny de l'arquitectura i el funcionament del sistema escollit; i tercer, la implementació d'aquest disseny escollit i la millora d'aquest tot el possible, tant en temps d'execució com en resultats.

- **CCO1.1: Avaluar la complexitat computacional d'un problema, conèixer estratègies algorísmiques que puguin dur a la seva resolució, i recomanar, desenvolupar i implementar la que garanteixi el millor rendiment d'acord amb els requisits establerts.** [Bastant].

  Aquesta segona competència, tot i que amb menys pes que l'anterior, també es treballa força al projecte. Juntament amb l'aplicació de CBR, s'ha d'implementar l'algorisme necessari per a la cerca dels k camins òptims dins del graf de la base de dades. Aquest s'ha d'escollir d'entre tots els que es puguin trobar a publicacions

relacionades per finalment adaptar-se a la situació concreta del problema i portar-se a terme la seva implementació. A més a més, també es recomanen i s'implementen altres estratègies algorísmiques com la resolució del problema amb relacions Específiques (realitzant el graf alternatiu) o la cerca de les combinacions finals mitjançant l'ordenació dels trajectes trobats entre les ciutats.